

core
Internet-Draft
Intended status: Standards Track
Expires: August 13, 2016

A. Bierman
YumaWorks
P. van der Stok
consultant
February 10, 2016

YANG Hash
draft-bierman-core-yang-hash-00

Abstract

This document describes an encoding of YANG names to 30 bit hashes. This document extends the CoMI draft to reduce payload and URI of CoMI network requests. The technique can be applied to other YANG based applications to reduce the payload by replacing the YANG names with 30 bit numbers.

Note

Discussion and suggestions for improvement are requested, and should be sent to core@ietf.org.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 13, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	3
1.1.1. Tree Diagrams	4
2. YANG Hash Generation	4
3. Re-Hash Error Procedure	5
4. Re-Hashing Names Procedure	6
5. ietf-yang-hash YANG Module	7
6. YANG Re-Hash Examples	9
6.1. Multiple Modules	11
6.2. Same Module	11
7. Retrieval of Rehashed Data	12
8. YANG Hash representations	13
8.1. YANG Hash in payload	13
8.2. YANG Hash in URL	13
9. YANG Hash Examples	14
10. Security Considerations	16
11. IANA Considerations	16
12. Acknowledgements	16
13. Changelog	17
14. References	17
14.1. Normative References	17
14.2. Informative References	18
Appendix A. Hash clash probability	18
Appendix B. Hash clash storage overhead	21
B.1. Server tables	21
B.2. Client tables	22
B.3. Table summary	22
Appendix C. payload reduction	23
Authors' Addresses	27

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. The messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

The drafts [I-D.ietf-netconf-restconf] and [I-D.vanderstok-core-comi] describe REST-like interfaces to access structured data defined in YANG [RFC6020].

The payload format CBOR [RFC7049] can be used to reduce the size of the transported payload. In that case the size of the payload depends for a large part on the YANG names. Reducing the names significantly reduces the payload size further (see Appendix C). This draft proposes a hashing technique to encode the YANG names into 30 bit numbers.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Readers of this specification should be familiar with all the terms and concepts discussed in [RFC2578].

The following terms are defined in the NETCONF protocol [RFC6241]: client, configuration data, data-store, and server.

The following terms are defined in the YANG data modelling language [RFC6020]: container, data node, key, key leaf, leaf, leaf-list, and list.

The following terms are defined in RESTCONF protocol [I-D.ietf-netconf-restconf]: data resource, data-store resource, edit operation, query parameter, target resource, and unified data-store.

The following terms are defined in this document:

YANG hash: CoMI object identifier, which is a 30-bit numeric hash of the YANG object identifier string for the object.

Rehash bit: Bit 31. If a particular YANG hash value is a re-hash for an identifier, then the rehash bit will be set in the object identifier. This allows the server to return descendant nodes that have been rehashed, instead of returning an error for an entire GET request.

Data-node instance: An instance of a data-node specified in a YANG module present in the server. The instance is stored in the memory of the server.

Notification-node instance: An instance of a schema node of type notification, specified in a YANG module present in the server.

The instance is generated in the server at the occurrence of the corresponding event and appended to a stream.

1.1.1. Tree Diagrams

A simplified graphical representation of the data model is used in the YANG modules specified in this document. The meaning of the symbols in these diagrams is as follows:

Brackets "[" and "]" enclose list keys.

Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

2. YANG Hash Generation

The association between string value and string number is done through a hash algorithm. The 30 least significant bits of the "murmur3" 32-bit hash algorithm are used. This hash algorithm is described online at [murmur3]. Implementations are available online [murmur-imp]. When converting 4 input bytes to a 32-bit integer in the hash algorithm, the Little-Endian convention MUST be used.

The "murmur3_32" hash function is executed for the entire path string. The value '42' is used as the seed for the hash function. The YANG hash is subsequently calculated by taking the 30 least significant bits.

The resulting 30-bit number is used by the server, unless the value is already being used for a different object by the server. In this case, the re-hash procedure in Section 3 is executed.

The hash is generated for the string representing the object path identifier. A canonical representation of the path identifier is used.

The module name is used to identify the namespace of the object node. The prefix cannot be used because it is allowed to change over time. The module name is never allowed to change.

The module name **MUST** be present in the identifier for the first node in the object path identifier.

If a child node in the object path identifier is from the same module namespace as its parent, then the module-name **MUST NOT** be used in the identifier.

If a child node in the object path identifier is not from the same module namespace as its parent, then the module-name **MUST** be used in the identifier.

Choice and case node names are not included in the path expression. Only 'container', 'list', 'leaf', 'leaf-list', and 'anyxml' nodes are listed in the path expression.

The YANG Hash value is calculated for all data nodes in the module, even if the server only implements a subset of these objects. This includes all "data-def", "rpc", "notification", and external data nodes derived from "augment" statements.

Example: the following canonical identifier is used for the 'mtu' leaf in the ietf-interfaces module:

```
/ietf-interfaces:interfaces/interface/mtu
```

Example: the following canonical identifier is used for the 'ipv4' container in the ietf-ip module, which augments the 'interface' list in the ietf-interfaces module:

```
/ietf-interfaces:interfaces/interface/ietf-ip:ipv4
```

3. Re-Hash Error Procedure

In most cases, the hash function is expected to produce unique values for all the node names supported by a constrained server. Given a known set of YANG modules, both server and client can calculate the YANG hashes independently, and offline.

Even though collisions are expected to happen rather rarely, they need to be considered (see Appendix A for clash probabilities). Collisions can be detected before deployment, if the vendor knows which modules are supported by the server, and hence all YANG hashes can be calculated. Collisions occur at a given server dependent on the set of modules supported by the server. The client needs to discover any re-hash mappings on a per server basis.

If the server needs to re-hash any YANG name, then it MUST create a "rehash" entry for all its rehashed node names, as described in Section 4.

A re-hashed object identifier has the rehash bit set in the identifier, every time it is sent from the server to the client. This allows the client to identify nodes for which a "reverse rehash" entry may need to be retrieved (see Section 6). A client does not need to retrieve the rehash map before retrieving or configuring rehashed data nodes.

If any node identifier provided by the client is not available because it has been rehashed, the server MUST return a rehash error, containing the 'rehash' entries for all the invalid nodes which were specified by the client.

It is possible that none of the node identifiers provided by the client in a GET method are invalid and rehashed, but rather one or more descendant nodes within the selected subtree(s) have been rehashed. In this case, a rehash error is not returned. Instead the requested subtree(s) are returned, and the rehash bit is set for any descendant node(s) that have been rehashed. The client will strip off the rehash bit and retrieve the 'revhash' entry for these nodes (if not already done).

4. Re-Hashing Names Procedure

A hash collision occurs if two different path identifier strings have the same hash value. If the server has around 1000 node names in its YANG modules, then the probability of a collision is a half per mil (see Appendix A). If a hash collision occurs on the server, then the node name that is causing the conflict has to be altered, such that the new hash value does not conflict with any value already in use by the server.

For example, rehashing could be done by prefixing a "~" character in front of the clashing name and execute murmur3 on the thus modified name. If necessary the prefixing can be done multiple times until the clashes are resolved. Using the prefixing is not needed from an inter-operability point of view but provides a procedure for client and server to calculate the rehash without reading the "rehash" entry.

5. ietf-yang-hash YANG Module

The "ietf-yang-hash" YANG module is used by the server to report any objects that have been mapped to produce a new hash value that does not conflict with any other YANG hash values used by the server.

YANG tree diagram for "ietf-yang-hash" module:

```
+--ro yang-hash
  +--ro rehash* [hash]
    +--ro hash      uint32
    +--ro object*
      +--ro module   string
      +--ro newhash  uint32
      +--ro path?    string
```

<CODE BEGINS> file "ietf-yang-hash@2016-02-10.yang"

```
module ietf-yang-hash {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-hash";
  prefix "yh";

  organization
    "IETF CORE (Constrained RESTful Environments) Working Group";

  contact
    "WG Web:  <http://tools.ietf.org/wg/core/>
    WG List:  <mailto:core@ietf.org>

    WG Chair: Carsten Bormann
               <mailto:cabo@tzi.org>

    WG Chair: Andrew McGregor
               <mailto:andrewmcgr@google.com>

    Editor:   Peter van der Stok
               <mailto:consultancy@vanderstok.org>

    Editor:   Andy Bierman
               <mailto:andy@yumaworks.com>

  description
    "This module contains re-hash information for the CoMI protocol.

    Copyright (c) 2016 IETF Trust and the persons identified as
```

authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices."

```
// RFC Ed.: replace XXXX with actual RFC number and remove this
// note.

// RFC Ed.: remove this note
// Note: extracted from draft-bierman-core-yang-hash-00.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2016-02-10 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: YANG Hash.";
}

container yang-hash {
  config false;
  description
    "Contains information on the YANG Hash values used by
    the server.";

  list rehash {
    key hash;
    description
      "Each entry describes an re-hash mapping in use by
      the server.";

    leaf hash {
      type uint32;
      description
        "The hash value that has a collision. This hash value
        cannot be used on the server. The rehashed
        value for each affected object must be used instead.";
    }

    list object {
```



```

min-elements 2;

description
    "Each entry identifies one of the objects involved in the
    hash collision and contains the rehash information for
    that object.";

leaf module {
    type string;
    mandatory true;
    description
        "The module name identifying the module namespace
        for this object.";
}

leaf newhash {
    type uint32;
    mandatory true;
    description
        "The new hash value for this object. The rehash bit is
        not set in this value.";
}

leaf path {
    type string;
    description
        "The object path identifier string used in the original
        YANG hash calculation. This object MUST be included for
        any objects in the rehash entry with the same 'module'
        value.";
}
}
}
}

<CODE ENDS>

```

6. YANG Re-Hash Examples

In this example there are two YANG modules, "foo" and "bar".

```
module foo {
  namespace "http://example.com/ns/foo";
  prefix "f";
```

```
revision 2015-06-07;

container A {
  list B {
    key name;
    leaf name { type string; }
    leaf counter1 { type uint32; }
  }
}

module bar {
  namespace "http://example.com/ns/bar";
  prefix "b";
  import foo { prefix f; }
  revision 2015-06-07;

  augment /f:A/f:B {
    leaf counter2 { type uint32; }
  }
}
```

This set of 3 YANG modules containing a total of 7 objects produces the following object list. Note that actual hash values are not shown, since these modules do not actually cause the YANG Hash clashes described in the examples.

Object	Path	Hash
foo:		
container	/foo:A	h1
list	/foo:A/B	h2
leaf	/foo:A/B/name	h3
leaf	/foo:A/B/counter1	h4
bar:		
leaf	/foo:A/B/bar1:counter2	h5

6.1. Multiple Modules

In this example, assume that the 'B' and 'counter2' objects produce the same hash value, so 'h2' and 'h5' both have the same value (e.g. '1234'):

The client might retrieve an entry from the list "/foo:A/B", which would cause this subtree to be returned. Instead, the server will return a message with the resource type "core.mg.yang-hash", representing the "yang-hash" data structure. Only the entry for the requested identifier is returned, even if multiple 'rehash' list entries exist.

REQ: GET example.com/mg/h2?keys="entry2"

RES: 4.00 "Bad Request" (Content-Format: application/cbor)

```
{
  "ietf-yang-hash:yang-hash" : {
    "rehash" : [
      {
        "hash" : 1234,
        "object" : [
          {
            "module" : "foo",
            "newhash" : 5678
          },
          {
            "module" : "bar",
            "newhash" : 8182
          }
        ]
      }
    ]
  }
}
```

6.2. Same Module

In this example, assume that the 'B', 'counter1', and 'counter2' objects produce the same hash value, so 'h2', 'h4', and 'h5' objects all have the same value (e.g. '1234'):

The client might retrieve an entry from the list "/foo:A/B", which would cause this subtree to be returned. Instead, the server will return a message with the resource type "core.mg.yang-hash", representing the "yang-hash" data structure. Only the entry for the

requested identifier is returned, even if multiple 'rehash' list entries exist.

REQ: GET example.com/mg/h2?keys="entry2"

RES: 4.00 "Bad Request" (Content-Format: application/cbor)

```
{
  "ietf-yang-hash:yang-hash" : {
    "rehash" : [
      {
        "hash" : 1234,
        "object" : [
          {
            "module" : "foo",
            "newhash" : 5678,
            "path" : "/foo:A/B"
          },
          {
            "module" : "foo",
            "newhash" : 2134,
            "path" : "/foo:A/B/counter1"
          },
          {
            "module" : "bar",
            "newhash" : 8182,
            "path" : "/foo:A/B/bar:counter2"
          }
        ]
      }
    ]
  }
}
```

7. Retrieval of Rehashed Data

In this example, assume that the 'B', 'counter1', and 'counter2' objects produce the same hash value, so 'h2', 'h4', and 'h5' objects all have the same value (e.g. '1234'):

The client might retrieve the top-level container "/foo:A", which would cause this subtree to be returned. Since the identifier (h1) has not been re-hashed, the server will return the requested data. The new hashes for 'h2', 'h4', and 'h5' will be returned, except the rehash bit will be set for these identifiers.

The notation "R+" indicates that the rehash bit is set.

REQ: GET example.com/mg/h1

RES: 2.05 Content (Content-Format: application/cbor)

```
{
  h1 : {
    R+5678 : {
      { h3 : "entry1"}:
        {R+2134: 615,
         R+8182: 7},
      { h3 : "entry2"}:
        {R+2134: 491,
         R+8182: 26}
    }
  }
}
```

The client will notice that the rehash bit is set for 3 nodes. The client will need to retrieve the full "yang-hash" container at this point, if that has not already been done. The rehashed identifiers will be in "rehash" list, contained in the "newhash" leaf for the "object" list.

8. YANG Hash representations

YANG hashes are represented in two fashions.

8.1. YANG Hash in payload

When a YANG hash value is printed in the payload, error-path or other string, then the lowercase hexadecimal representation is used. Leading zeros are used so the value uses 8 hex characters.

8.2. YANG Hash in URL

When a URL contains a YANG hash, it is encoded using base64url "URL and Filename safe" encoding as specified in [RFC4648].

The hash H is represented as a 30-bit integer, divided into five 6-bit integers as follows:

```
B1 = (H & 0x3f000000) >> 24
B2 = (H & 0xfc0000) >> 18
B3 = (H & 0x03f000) >> 12
B4 = (H & 0x000fc0) >> 6
B5 = H & 0x00003f
```

Subsequently, each 6-bit integer B_x is translated into a character C_x using Table 2 from [RFC4648], and a string is formed by concatenating the characters in the order C₁, C₂, C₃, C₄, C₅.

For example, the YANG hash 0x29abdcca is encoded as "pq9zK".

9. YANG Hash Examples

The YANG hash value for 'current-datetime' is calculated by constructing the schema node identifier for the object:

```
/ietf-system:system-state/clock/current-datetime
```

The 30 bit murmur3 hash value (see Section 2) is calculated on this string with hash: 0x047c468b and EfEaM. The request using this hash value is shown below:

```
REQ: GET example.com/mg/EfEaM
```

```
RES: 2.05 Content (Content-Format: application/cbor)
{
  0x047c468b : "2014-10-26T12:16:31Z"
}
```

The YANG hash values for 'clock', 'current-datetime', and 'boot-datetime' are calculated by constructing the schema node identifier for the objects, and then calculating the 30 bit murmur3 hash values (shown in parenthesis):

```
/ietf-system:system-state/clock (0x021ca491 and CDKSQ)
/ietf-system:system-state/clock/current-datetime (0x047c468b)
/ietf-system:system-state/clock/boot-datetime (0x1fb5f4f8)
```

The YANG hash values for 'neighbor', 'ip', and 'link-layer-address' are calculated by constructing the schema node identifier for the objects, and then calculating the 30 bit murmur3 hash values (shown in parenthesis):

```
/ietf-interfaces:interfaces/interface/ietf-ip:ipv6/neighbor
  (0x2445e478 and kReR4)
/ietf-interfaces:interfaces/interface/ietf-ip:ipv6/neighbor/ip
  (0x2283ed40 and ig-la)
/ietf-interfaces:interfaces/interface/ietf-ip:ipv6/neighbor/
  link-layer-address (0x3d6915c7)
```

The YANG translation of the SMI specifying the ipNetToMediaTable [RFC4293] yields:

```
container IP-MIB {
  container ipNetToPhysicalTable {
    list ipNetToPhysicalEntry {
      key "ipNetToPhysicalIfIndex
          ipNetToPhysicalNetAddressType
          ipNetToPhysicalNetAddress";
      leaf ipNetToMediaIfIndex {
        type: int32;
      }
      leaf ipNetToPhysicalIfIndex {
        type if-mib:InterfaceIndex;
      }
      leaf ipNetToPhysicalNetAddressType {
        type inet-address:InetAddressType;
      }
      leaf ipNetToPhysicalNetAddress {
        type inet-address:InetAddress;
      }
      leaf ipNetToPhysicalPhysAddress {
        type yang:phys-address {
          length "0..65535";
        }
      }
      leaf ipNetToPhysicalLastUpdated {
        type yang:timestamp;
      }
      leaf ipNetToPhysicalType {
        type enumeration { ... }
      }
      leaf ipNetToPhysicalState {
        type enumeration { ... }
      }
      leaf ipNetToPhysicalRowStatus {
        type snmpv2-tc:RowStatus;
      }
    }
  }
}
```

The YANG hash values for 'ipNetToPhysicalEntry' and its child nodes are calculated by constructing the schema node identifier for the objects, and then calculating the 30 bit murmur3 hash values (shown in parenthesis):

```
/IP-MIB:IP-MIB/ipNetToPhysicalTable (0x0aba15cc and kuhXM)
/IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry
  (0xo6aaddbc and Gqt28)
/IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/
  ipNetToPhysicalIfIndex (0x346b3071)
```

```
/IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/  
  ipNetToPhysicalNetAddressType (0x3650bb64)  
/IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/  
  ipNetToPhysicalNetAddress (0x06fd4d91)  
/IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/  
  ipNetToPhysicalPhysAddress (0x26180bcb)  
/IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/  
  ipNetToPhysicalLastUpdated (0x3d6bbe90)  
/IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/  
  ipNetToPhysicalType (0x35ecbb3d)  
/IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/  
  ipNetToPhysicalState (0x13038bb5)  
/IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/  
  ipNetToPhysicalRowStatus (0x09e1fa37)
```

The YANG Hash values for the YANG Patch request objects are calculated as follows:

```
0x2c3f93c7: /ietf-yang-patch:yang-patch  
0x2fb8873e: /ietf-yang-patch:yang-patch/patch-id  
0x011640f0: /ietf-yang-patch:yang-patch/comment  
0x16804b72: /ietf-yang-patch:yang-patch/edit  
0x2bd93228: /ietf-yang-patch:yang-patch/edit/edit-id  
0x1959d8c9: /ietf-yang-patch:yang-patch/edit/operation  
0x1346e0aa: /ietf-yang-patch:yang-patch/edit/target  
0x0750e196: /ietf-yang-patch:yang-patch/edit/point  
0x0b45277e: /ietf-yang-patch:yang-patch/edit/where  
0x2822c407: /ietf-yang-patch:yang-patch/edit/value
```

10. Security Considerations

The replacement of name-strings by numbers does not affect the security of the transmitted requests.

11. IANA Considerations

No considerations for IANA apply.

12. Acknowledgements

We are very grateful to Bert Greevenbosch who suggested the use of hashes and specified the use of murmur3. Many thanks for their contributions go to Alexander Pelov, Juergen Schonwalder, Anuj Sehgal and Michel Veillette.

This material is based upon work supported by the The Space & Terrestrial Communications Directorate (S&TCD) under Contract No. W15P7T-13-C-A616. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of The Space & Terrestrial Communications Directorate (S&TCD) .

13. Changelog

Version 0 is extracted from comi draft version 8 [I-D.vanderstok-core-comi]. Changed Appendix A, and added Appendix C.

14. References

14.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [murmur3] , "murmurhash family", Web <http://en.wikipedia.org/wiki/MurmurHash>, .
- [murmur-imp] , "murmurhash implementation", Web <https://code.google.com/p/smhasher/>, .

14.2. Informative References

- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <<http://www.rfc-editor.org/info/rfc2578>>.
- [RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <<http://www.rfc-editor.org/info/rfc4293>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [I-D.ietf-netconf-restconf] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-07 (work in progress), July 2015.
- [I-D.vanderstok-core-comi] Stok, P., Bierman, A., Schoenwaelder, J., and A. Sehgal, "CoAP Management Interface", draft-vanderstok-core-comi-08 (work in progress), October 2015.
- [coll-prob] Presching, j., "Hash collision probabilities", Web <http://presching.com/20110504/hash-collision-probabilities>, May 2011.
- [birthday] Wikipedia, , "Birthday problem", Web https://en.wikipedia.org/wiki/Birthday_problem, .

Appendix A. Hash clash probability

Number of names	28 bits	29 bits	30 bits	31 bits	32 bits	33 bits
10	1,7E-07	8,4E-08	4,2E-08	2,1E-08	1,1E-08	5,2E-09
100	1,8E-05	9,2E-06	4,6E-06	2,3E-06	1,2E-06	5,8E-07

200	7,4E-05	3,7E-05	1,9E-05	9,3E-06	4,6E-06	2,3E-06
10 ³	1,9E-03	9,3E-04	4,7E-04	2,3E-04	1,2E-04	5,8E-05
4000	3,0E-02	1,5E-02	7,5E-03	3,7E-03	1,9E-03	9,3E-04
10 ⁴	1,9E-01	9,3E-02	4,6E-02	2,3E-02	1,2E-02	5,8E-03

Table 1: Probability of one or more clashes

This appendix calculates the probability of a hash clash as function of the hash size and the number of YANG names. The standard way to calculate the probability of a clash is to calculate the probability that no clashes occur [birthday], [coll-prob].

The probability of no clashes when generating k numbers with a hash size of $N=2^{\text{bits}}$ is given by:

$$D(N,k) = ((N-1)/N) * ((N-2)/N) * \dots * (N-(k-1))/N \quad (1)$$

which can be approximated with:

$$\exp(-k*(k-1)/2N) \quad (2)$$

The probability that one or more clashes occur is given by:

$$1 - \exp(-k*(k-1)/2N) \sim k*(k-1)/2N \quad (3)$$

Table 1 shows the probabilities for a given set of values of $N=2^{\text{bits}}$ and number of YANG node names k. Probabilities which are larger than 0.5 are not shown because the used approximations are not accurate any more.

The overhead in servers and clients depends on the number of clashes. Therefore it is interesting to know the probability that more than one clash occurs. The probability of generating k numbers with a hash size of $N=2^{\text{bits}}$, where 2 numbers are identical and all the rest is different, is composed of the following parts. The probability that the second number is equal to the first is $1/N$. The possible number of configurations of 2 equal numbers out of k is given by $\sum_{i=1, k-1} (i)$. The probability of $k-1$ different numbers is given by $D(N, k-1)$. The probability of generating exactly one clash of two numbers is given by:

$$(\sum_{i=1, k-1} (i)) * D(N, k-1) / N$$

Where we used formula (1). Working out the summation and using (2), the probability that exactly one pair of hashes clashes is given by:

$$(k * (k-1) / 2N) * \exp(-(k-1) * (k-2) / 2N)$$

The probability that more than one pair clashes is given by the probability that a clash occurs minus the probability that only one pair clashes. This leads to:

$$1 - \exp(-(k * (k-1) / 2N)) - (k * (k-1) / 2N) * \exp(-(k-1) * (k-2) / 2N)$$

Substituting formula 3, gives:

$$k * (k-1) / 2N - k * (k-1) / 2N + (k * (k-1) ^2 * (k-2) / 4N^2 =$$

$$(k * (k-1) ^2 * (k-2) / 4N^2$$

Number of names	28 bits	29 bits	30 bits	31 bits	32 bits	33 bits
10	2,3E-14	5,6E-15	1,4E-15	3,5E-16	8,8E-17	2,2E-17
100	3,3E-10	8,3E-11	2,1E-11	5,2E-12	1,3E-12	3,3E-13
200	5,4E-09	1,4E-09	3,4E-10	8,5E-11	2,1E-11	5,3E-12
10^3	3,5E-06	8,6E-07	2,2E-07	5,4E-08	1,4E-08	3,4E-09
4000	8,9E-04	2,2E-04	5,6E-05	1,4E-05	3,5E-06	8,7E-07

10 ⁴	3,5E-02	8,7E-03	2,2E-03	5,4E-04	1,4E-04	7 3,4E-0 5
-----------------	---------	---------	---------	---------	---------	------------------

Table 2: Probability of more than 2 entries equal clashes

The corresponding probabilities are shown in Table 2. Assuming a hash size of 2^{30} , and about 1000 YANG nodes in a server, the probability of one clashing pair is $0.5 \cdot 10^{-3}$, and the probability that more clashes occur is $2 \cdot 10^{-7}$.

Appendix B. Hash clash storage overhead

Clashes may occur in servers dynamically during the operation of their clients, and clashes must be handled on a per server basis in the client. When rehashing is possible, clashing names on a given server are prefixed with a character (for example "~") and are rehashed, thus leading to hash values which uniquely identify the data nodes in the server. This appendix calculates the storage space needed when a clash occurs in a set of servers running the same server code. Appendix A shows that more than one clash in a server set is exceptional, which suggests at most two clashing object names in a given server.

The sizes of server and client tables needed to handle the clashes in client and server are calculated separately, because they differ significantly.

B.1. Server tables

When a request arrives at the server, the server must relate the incoming hash value to the memory locations where the related values are stored. In the server a translation table must be provided that relates a hash value to a memory address where either the raw data or a description of the data (as prescribed by the YANG compiler) are stored. The required storage space is a sequence of (32 bit yang hash, 64 bit memory address) for every YANG data node. The translation table size in a server is 12 bytes times the number of YANG data nodes in the server.

For every clashing hash value the following server clash table entries are needed: Clashed hash value, module name, and new hash. To reduce table size in the client, module name can be replaced with a 1 byte module identifier. The module identifier represents the index value of an array of module names. Server clash table size is: 2 hashes (8 bytes) + 1 module identifier (1 byte)

B.2. Client tables

In the client, the compiled code must refer to a hash value. To cope with on-the-fly rehashing, the compiled code needs to invoke a procedure that returns the possibly rehashed value as function of the original hash value, module name, and server address. The client needs to store a client clash table containing: the clashed hash value, module name, server IPv6 address (or name), and rehash value for as many rehashes occurring in a given server. Many servers contain an identical set of YANG modules. The servers containing the same module set belong to the same server type. The server type is used to administrate the hash clash occurrence. To reduce client clash table size, module name can be replaced with a 1 byte module identifier. The module identifier represents the index value of an array of module names. A table of IPv6 server addresses must already exist in the client. To reduce client clash table size further, the server IPv6 address can be replaced with a 1 byte server type identifier. The server table can be ordered according to server type. A table with server type and pointer to sub-table start suffices to find all IPv6 addresses belonging to a server type.

The client clash table reduces to clashed hash value (4 bytes), module identifier (1 byte), server type identifier (1 byte) and rehash value (4 bytes).

B.3. Table summary

Sizes of all the tables are:

Server clash table: 9 bytes per clashing object name.

Client clash table: 10 bytes per server type, per clashing object name.

Array of module names: Sum of module name sizes.

Server identifier table: 1 byte server type + 4 bytes pointer per server type.

The existence of the translation table in a server is required independent of rehashing. The table sizes calculated to estimate the storage requirements coming from CoMI clashes. Assume the following numbers:

- o 500 data nodes per server
- o 10 server types

- o 30 modules
- o Module name is on average 20 bytes
- o Maximum of 2 clashing object names occurring in 2 server types

This yields the following overhead estimates:

Server tables size:

- * Server clash table: 2×9 bytes represents 18 bytes
- * Module name array: 30×20 represents 600 bytes

Client table sizes:

- * Client clash table: $10 \times 2 \times 2$ represents 40 bytes for 2 object names in 2 server types.
- * Module name array: 30×20 represents 600 bytes.
- * Server identifier table: $10 \times 5 = 50$ bytes

In conclusion:

1. Storage space size in client is independent of number of servers but depends on number of server types.
2. There is a common storage size for the module array of 600 bytes.
3. Assuming 2 clashing object names in 2 server types, additional storage space in client is 40 bytes and in server 18 bytes.
4. When the module array is suppressed (removing 600 bytes storage space), the server clash table and the client clash table increase with 40 bytes and 80 bytes respectively.

Appendix C. payload reduction

The hashing of the YANG identifier in the transported payload reduces the size of the YANG objects transported in the payload. This note calculates the payload size reduction and the number of YANG objects that can be transported in a single 802.154 CoAP packet. The payload is assumed to be a sequence of maps, where the entry ("ident" : value) is referred to as a map, as shown below. The value can be an integer or a string.

```
{  
  "ident1" : value 1,  
  "ident2" : value 2.  
  Etc .....  
}
```

In general, we can assume that the payload size (SI) of a YANG identifier string lies between 6 to 128 bytes with an average of 30-40 bytes. The payload size (SV) of a value can range between 1 byte to 128 bytes. The transport format is CBOR. The overhead coming from CBOR is composed of:

- o One byte to indicate the number of maps (> 2 maps in the example above).
- o Two bytes per map: one describing the identifier, and one describing the value.

When the CBOR byte describes an integer (major type 0), the size of the following integer is 0, when integer < 24. Otherwise the size of the following integer is 1 to 8 bytes. The size of the string remains unchanged when preceded by a CBOR byte (major type 2). When the string size < 24, no extra bytes are needed; when the string size lies between 24 and 128 one extra CBOR byte overhead is needed. The parameters SE, SV and SI are introduced with the following meaning:

- o SE is the size of the identifier encoded by a hash or other unsigned integer, with $0 < SE < 5$; because the hash size is 4 bytes and the minimum managed encoded identifier is assumed to take 1 byte.
- o SV is the size of the value, with $0 \leq SV < 128$.
- o SI is the size of the original YANG hash identifier string, with $4 < SI < 64$.

The impact of the conversion from YANG identifier string to unsigned integer is straightforward to calculate per map. It is assumed that one CBOR byte or two CBOR bytes are needed dependent on the sizes SI and SV. A map size with the original YANG identifier string is given by:

- o Map size is: $2 + SI + SV$, when $SI, SV < 24$;
- o Map size is: $3 + SI + SV$, when $SI < 24$ and $SV > 23$, or $SI > 23$ and $SV < 24$;

- o Map size is: $4 + SI + SV$, when $SI, SV > 23$.

The map size when SI is converted to an unsigned integer with size SE is given by:

- o Encoded map size is: $2 + SE + SV$, when $SV < 24$;
- o Encoded map size is: $3 + SE + SV$, when $SV > 23$.

The improvement of the conversion can be written as

- o $(2+SE+SV)/(2+SI+SV)$ when $SI, SV < 24$;
- o $(2+SE+SV)(3+SI+SV)$ when $SI > 23$ and $SV < 24$;
- o $(3+SE+SV)/(3+SI+SV)$ when $SI < 24$, and $SV > 23$;
- o $(3+SE+SV)/(4+SI+SV)$ when $SI, SV > 23$.

Table 3 shows the size reduction for different SI and SV values and $SE = 4$:

SV->	0	4	10	20	30	40	60
SI=6	0,75	0,83	0,89	0,93	0,95	0,96	0,97
SI=10	0,83	0,88	0,91	0,94	0,95	0,96	0,97
SI=20	0,91	0,92	0,94	0,95	0,96	0,97	0,98
SI=30	0,97	0,97	0,98	0,98	0,98	0,99	0,99
SI=40	0,98	0,98	0,98	0,98	0,99	0,99	0,99

Table 3: Payload size reduction as function of SI and SV

Another way to look at it is to see how many maps fit in a packet. Assuming a 802.15.4 packet with short addresses, the IEEE header is 13 bytes. The CoAP header assuming only mesh traffic takes 6 bytes. The URI is composed of 3 options, where each option header takes 1 byte: total of 3 bytes. The URI is assumed to be split up in the following 3 parts:

- o URI-host "example.net" takes 13 bytes, which necessitates 1 length byte: total of 14 bytes.

- o URI-path = "mg" takes 4 bytes.
- o URI-path = "hash value" takes 7 bytes.

Total URI takes $3+14+4+7 = 28$ bytes. For the CoMI payload, there remains $127 - 13 - 6 - 28 = 80$ bytes. Subtracting the byte indicating the number of CBOR maps, the payload size for the maps is 79 bytes.

N.B. no security overhead is included.

When the YANG string identifier needs to be stored, the number of storable maps is given by:

- o $79/(2+SI+SV)$ for $SI, SV < 24$;
- o $79/(3+SI+SV)$ for $SI < 24$ and $SV > 23$, or $SI > 23$ and $SV < 24$;
- o $79/(4+SI+SV)$ for $SI, SV > 23$.

Table 4 shows the number of transported maps for different values of SI and SV.

SV->	0	4	10	20	30	40	60
SI=6	9	6	4	2	2	1	1
SI=10	6	4	3	2	1	1	1
SI=20	3	3	2	1	1	1	0
SI=30	2	2	1	1	1	1	0
SI=40	1	1	1	1	1	0	0

Table 4: Nr of transported maps as function of SI and SV

Assuming the encoding of the YANG identifier to an unsigned integer with size $0 < SE < 5$, the number of storable maps is given by:

- o $79/(2+SE+SV)$ for $SV < 24$;
- o $79/(3+SE+SV)$ for $SV > 23$.

Table 5 shows the number of transported maps for different values of SE and SV, independent of SI.

SV->	0	4	10	20	30	40	60
SE=1	26	11	6	3	2	1	1
SE=2	19	9	5	3	2	1	1
SE=3	15	8	5	3	2	1	1
SE=4	13	7	4	3	2	1	1

Table 5: Nr of transported maps as function of SE and SV,

The value of SI for the average YANG string identifier is 30. When the size of the value part of the map is less than 10 ($SV < 10$) and $SI=30$, the impact of the hashing is significant: 10 maps can be transported instead of 1 or 2. Further reducing the value of SE from 4 to 1 increases the number of transported maps with a factor 2 to 1,2.

Authors' Addresses

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 29, 2016

C. Bormann
Universitaet Bremen TZI
November 26, 2015

Block-wise transfers in CoAP: Extension for Reliable Transport (BERT)
draft-bormann-core-block-bert-00

Abstract

CoAP (RFC7252) is a RESTful transfer protocol for constrained nodes and networks, originally using UDP or DTLS over UDP as its transport. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. CoAP's Block protocol (draft-ietf-core-block) allows transferring larger payloads over limited-size datagrams -- for instance, for firmware updates.

CoAP over TCP and TLS (draft-ietf-core-tcp-tls) enables the use of extended, but not unlimited, size messages. The present specification, Block-wise transfers in CoAP: Extension for Reliable Transport (BERT), extends the block protocol in a simple way to be able to make use of these larger messages over a reliable transport.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 29, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Objectives	2
1.2. Terminology	3
2. BERT Blocks	3
2.1. Caching Considerations	4
2.2. Open Questions	4
2.3. Combining BERT blocks with the Observe Option	4
3. Examples	4
3.1. Block2 Example	5
3.2. Block1 Example	5
4. IANA Considerations	6
5. Security Considerations	6
6. Acknowledgements	6
7. References	6
7.1. Normative References	6
7.2. Informative References	6
Author's Address	7

1. Introduction

(see abstract for now)

1.1. Objectives

The objectives stated in the introduction of [I-D.ietf-core-block] apply to the present document as well. (The exception is the desire to enable individual retransmissions -- this is already handled by reliable transport.)

Specifically, this specification continues to minimize the need for creation of additional state, even if a TCP (or TLS over TCP) connection already requires more state than a basic CoAP client-to-server relationship.

An important aspect of this also is the need for state at proxies, see Section 2.1.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119] and indicate requirement levels for compliant implementations.

In this document, the term "byte" is used in its now customary sense as a synonym for "octet".

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C, except that the operator "***" stands for exponentiation.

BERT Option:

A Block1 or Block2 option that includes an SZX value of 7.

BERT Block:

The payload of a CoAP message that is affected by a BERT Option in descriptive usage (Section 2.1 of [I-D.ietf-core-block]).

2. BERT Blocks

The use of the present extension is signalled by sending Block1 or Block2 options with SZX == 7 (a "BERT option"). (SZX == 7 is a value that was reserved in [I-D.ietf-core-block].)

In control usage, a BERT option is interpreted in the same way as the equivalent option with SZX == 6, except that it also indicates the capability to process BERT blocks. As with the basic Block protocol, the recipient of a CoAP request with a BERT option in control usage is allowed to respond with a different SZX value, e.g. to send a non-BERT block instead.

In descriptive usage, a BERT option is interpreted in the same way as the equivalent option with SZX == 6, except that the payload is allowed to contain a multiple of 1024 bytes (non-final BERT block) or more than 1024 bytes (final BERT block).

The recipient of a non-final BERT block (M=1) conceptually partitions the payload into a sequence of 1024-byte blocks and acts exactly as if it had received this sequence in conjunction with block numbers starting at, and sequentially increasing from, the block number given in the Block option. In other words, the entire BERT block is positioned at the byte position that results from multiplying the block number with 1024. The position of further blocks to be transferred is indicated by incrementing the block number by the

number of elements in this sequence (i.e., the size of the payload divided by 1024 bytes).

As with `SZX == 6`, the recipient of a final BERT block (`M=0`) simply appends the payload at the byte position that is indicated by the block number multiplied with 1024.

2.1. Caching Considerations

Section 2.10 of [I-D.ietf-core-block] applies unchanged.

Discussion: As with the basic Block protocol, a proxy may need to re-slice blocks. Requiring BERT blocks to start at 1024 byte boundaries simplifies this considerably.

2.2. Open Questions

Does the use of CoAP over TCP or TLS simply imply BERT capability or do we explicitly signal that? Signalling is easy for Block2 (but does require sending Block2 options with the value 7 as a matter of course), less so for Block1.

If an optimistic approach is desired, the error code 4.13 (Request Entity Too Large) could be employed as defined in Section 2.5 of [I-D.ietf-core-block].

2.3. Combining BERT blocks with the Observe Option

BERT Blocks combine with the Observe Option exactly as defined for basic blocks in Section 2.6 of [I-D.ietf-core-block].

3. Examples

This section extends Section 3 of [I-D.ietf-core-block] with a few examples that involve BERT options. Extending the notation used in that section, a value of `SZX == 7` is shown as "BERT", or as "BERT(nnn)" to indicate a payload of size nnn.

In all these examples, a Block option is shown in a decomposed way indicating the kind of Block option (1 or 2) followed by a colon, and then the block number (NUM), more bit (M), and block size exponent ($2^{*(SZX+4)}$) separated by slashes. E.g., a Block2 Option value of 33 would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

3.1. Block2 Example

The first example (Figure 1) shows a GET request with a response that is split into three BERT blocks. The first response contains 3072 bytes of payload; the second, 5120; and the third, 4711. Note how the block number increments to move the position inside the response body forward.

CLIENT	SERVER
GET, /status	----->
<----- 2.05 Content, 2:0/1/BERT(3072)	
GET, /status, 2:3/0/BERT	----->
<----- 2.05 Content, 2:3/1/BERT(5120)	
GET, /status, 2:8/0/BERT	----->
<----- 2.05 Content, 2:8/0/BERT(4711)	

Figure 1: GET with BERT blocks

3.2. Block1 Example

The following example (Figure 2) demonstrates a PUT exchange with BERT blocks.

CLIENT	SERVER
PUT, /options, 1:0/1/BERT(8192)	----->
<----- 2.31 Continue, 1:0/1/BERT	
PUT, /options, 1:8/1/BERT(16384)	----->
<----- 2.31 Continue, 1:8/1/BERT	
PUT, /options, 1:24/0/BERT(5683)	----->
<----- 2.04 Changed, 1:24/0/BERT	

Figure 2: PUT with BERT blocks

4. IANA Considerations

This specification makes no requests of IANA.

(This section to be removed by the RFC editor.)

5. Security Considerations

The Security Considerations of [I-D.ietf-core-block] apply unchanged.

6. Acknowledgements

7. References

7.1. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP",
draft-ietf-core-block-18 (work in progress), September
2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
Application Protocol (CoAP)", RFC 7252, DOI 10.17487/
RFC7252, June 2014,
<<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained
Application Protocol (CoAP)", RFC 7641, DOI 10.17487/
RFC7641, September 2015,
<<http://www.rfc-editor.org/info/rfc7641>>.

7.2. Informative References

- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Technologies, Z., and H.
Tschafenig, "A TCP and TLS Transport for the Constrained
Application Protocol (CoAP)", draft-ietf-core-coap-tcp-
tls-01 (work in progress), November 2015.

- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <<http://www.rfc-editor.org/info/rfc4919>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.

Author's Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 12, 2016

C. Bormann
Universitaet Bremen TZI
June 10, 2016

Block-wise transfers in CoAP: Extension for Reliable Transport (BERT)
draft-bormann-core-block-bert-01

Abstract

CoAP (RFC7252) is a RESTful transfer protocol for constrained nodes and networks, originally using UDP or DTLS over UDP as its transport. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. CoAP's Block protocol (draft-ietf-core-block) allows transferring larger payloads over limited-size datagrams -- for instance, for firmware updates.

CoAP over TCP and TLS (draft-ietf-core-tcp-tls) enables the use of extended, but not unlimited, size messages. The present specification, Block-wise transfers in CoAP: Extension for Reliable Transport (BERT), extends the block protocol in a simple way to be able to make use of these larger messages over a reliable transport.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 12, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Objectives	2
1.2. Terminology	3
2. BERT Blocks	3
2.1. Caching Considerations	4
2.2. Open Questions	4
2.3. Combining BERT blocks with the Observe Option	4
3. Examples	4
3.1. Block2 Example	5
3.2. Block1 Example	5
4. IANA Considerations	6
5. Security Considerations	6
6. Acknowledgements	6
7. References	6
7.1. Normative References	6
7.2. Informative References	6
Author's Address	6

1. Introduction

(see abstract for now)

1.1. Objectives

The content of this document is intended for integration into [I-D.ietf-core-coap-tcp-tls].

The objectives stated in the introduction of [I-D.ietf-core-block] apply to the present document as well. (The exception is the desire to enable individual retransmissions -- this is already handled by reliable transport.)

Specifically, this specification continues to minimize the need for creation of additional state, even if a TCP (or TLS over TCP) connection already requires more state than a basic CoAP client-to-server relationship.

An important aspect of this also is the need for state at proxies, see Section 2.1.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119] and indicate requirement levels for compliant implementations.

The definitions of [RFC7252] apply.

In this document, the term "byte" is used in its now customary sense as a synonym for "octet".

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C, except that the operator "***" stands for exponentiation.

BERT Option:

A Block1 or Block2 option that includes an SZX value of 7.

BERT Block:

The payload of a CoAP message that is affected by a BERT Option in descriptive usage (Section 2.1 of [I-D.ietf-core-block]).

2. BERT Blocks

The use of the present extension is signalled by sending Block1 or Block2 options with SZX == 7 (a "BERT option"). (SZX == 7 is a value that was reserved in [I-D.ietf-core-block].)

In control usage, a BERT option is interpreted in the same way as the equivalent option with SZX == 6, except that it also indicates the capability to process BERT blocks. As with the basic Block protocol, the recipient of a CoAP request with a BERT option in control usage is allowed to respond with a different SZX value, e.g. to send a non-BERT block instead.

In descriptive usage, a BERT option is interpreted in the same way as the equivalent option with SZX == 6, except that the payload is allowed to contain a multiple of 1024 bytes (non-final BERT block) or more than 1024 bytes (final BERT block).

The recipient of a non-final BERT block (M=1) conceptually partitions the payload into a sequence of 1024-byte blocks and acts exactly as if it had received this sequence in conjunction with block numbers

starting at, and sequentially increasing from, the block number given in the Block option. In other words, the entire BERT block is positioned at the byte position that results from multiplying the block number with 1024. The position of further blocks to be transferred is indicated by incrementing the block number by the number of elements in this sequence (i.e., the size of the payload divided by 1024 bytes).

As with `SZX == 6`, the recipient of a final BERT block (`M=0`) simply appends the payload at the byte position that is indicated by the block number multiplied with 1024.

2.1. Caching Considerations

Section 2.10 of [I-D.ietf-core-block] applies unchanged.

Discussion: As with the basic Block protocol, a proxy may need to re-slice blocks. Requiring BERT blocks to start at 1024 byte boundaries simplifies this considerably.

2.2. Open Questions

Does the use of CoAP over TCP or TLS simply imply BERT capability or do we explicitly signal that? Signalling is easy for Block2 (but does require sending Block2 options with the value 7 as a matter of course), less so for Block1.

If an optimistic approach is desired, the error code 4.13 (Request Entity Too Large) could be employed as defined in Section 2.5 of [I-D.ietf-core-block].

2.3. Combining BERT blocks with the Observe Option

BERT Blocks combine with the Observe Option ([RFC7641] exactly as defined for basic blocks in Section 2.6 of [I-D.ietf-core-block].

3. Examples

This section extends Section 3 of [I-D.ietf-core-block] with a few examples that involve BERT options. Extending the notation used in that section, a value of `SZX == 7` is shown as "BERT", or as "BERT(nnn)" to indicate a payload of size nnn.

In all these examples, a Block option is shown in a decomposed way indicating the kind of Block option (1 or 2) followed by a colon, and then the block number (NUM), more bit (M), and block size exponent ($2^{*(SZX+4)}$) separated by slashes. E.g., a Block2 Option value of 33

would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

3.1. Block2 Example

The first example (Figure 1) shows a GET request with a response that is split into three BERT blocks. The first response contains 3072 bytes of payload; the second, 5120; and the third, 4711. Note how the block number increments to move the position inside the response body forward.

CLIENT	SERVER
GET, /status	----->
<----- 2.05 Content, 2:0/1/BERT(3072)	
GET, /status, 2:3/0/BERT	----->
<----- 2.05 Content, 2:3/1/BERT(5120)	
GET, /status, 2:8/0/BERT	----->
<----- 2.05 Content, 2:8/0/BERT(4711)	

Figure 1: GET with BERT blocks

3.2. Block1 Example

The following example (Figure 2) demonstrates a PUT exchange with BERT blocks.

CLIENT	SERVER
PUT, /options, 1:0/1/BERT(8192)	----->
<----- 2.31 Continue, 1:0/1/BERT	
PUT, /options, 1:8/1/BERT(16384)	----->
<----- 2.31 Continue, 1:8/1/BERT	
PUT, /options, 1:24/0/BERT(5683)	----->
<----- 2.04 Changed, 1:24/0/BERT	

Figure 2: PUT with BERT blocks

4. IANA Considerations

This specification makes no requests of IANA.

(This section to be removed by the RFC editor.)

5. Security Considerations

The Security Considerations of [I-D.ietf-core-block] apply unchanged.

6. Acknowledgements

7. References

7.1. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP",
draft-ietf-core-block-20 (work in progress), April 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
Application Protocol (CoAP)", RFC 7252,
DOI 10.17487/RFC7252, June 2014,
<<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained
Application Protocol (CoAP)", RFC 7641,
DOI 10.17487/RFC7641, September 2015,
<<http://www.rfc-editor.org/info/rfc7641>>.

7.2. Informative References

- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Technologies, Z., and H.
Tschofenig, "A TCP and TLS Transport for the Constrained
Application Protocol (CoAP)", draft-ietf-core-coap-tcp-
tls-02 (work in progress), April 2016.

Author's Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

C. Bormann
Universitaet Bremen TZI
A. Betzler
C. Gomez
I. Demirkol
Universitat Politecnica de Catalunya/Fundacio i2CAT
October 19, 2015

CoAP Simple Congestion Control/Advanced
draft-bormann-core-cocoa-03

Abstract

The CoAP protocol needs to be implemented in such a way that it does not cause persistent congestion on the network it uses. The CoRE CoAP specification defines basic behavior that exhibits low risk of congestion with minimal implementation requirements. It also leaves room for combining the base specification with advanced congestion control mechanisms with higher performance.

This specification defines some simple advanced CoRE Congestion Control mechanisms, Simple CoCoA. In the present version -02, it is making use of input from simulations and experiments in real networks. The specification might still benefit from simplifying it further.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Context	3
3. Area of Applicability	4
4. Advanced CoAP Congestion Control: RTO Estimation	4
4.1. Blind RTO Estimate	5
4.2. Measured RTO Estimate	5
4.2.1. Modifications to the algorithm of RFC 6298	5
4.2.2. Discussion	6
4.3. Lifetime, Aging	6
5. Advanced CoAP Congestion Control: Non-Confirmables	7
5.1. Discussion	7
6. Advanced CoAP Congestion Control: Aggregate Congestion Control	8
6.1. Proposed Algorithm	8
6.2. Example	8
6.3. Discussion	9
7. IANA Considerations	10
8. Security Considerations	10
9. Acknowledgements	10
10. References	10
10.1. Normative References	10
10.2. Informative References	11
Authors' Addresses	12

1. Introduction

(See Abstract.)

Extended rationale for this specification can be found in [I-D.bormann-core-congestion-control] and

[I-D.eggert-core-congestion-control], as well as in the minutes of the IETF 84 CoRE WG meetings.

1.1. Terminology

This specification uses terms from [RFC7252]. In addition, it defines the following terminology:

Initiator: The endpoint that sends the message that initiates an exchange. E.g., the party that sends a confirmable message, or a non-confirmable message conveying a request.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

(Note that this document is itself informational, but it is discussing normative statements.)

The term "byte", abbreviated by "B", is used in its now customary sense as a synonym for "octet".

2. Context

In the Vancouver IETF 84 CoRE meeting, a path forward was defined that includes a very simple basic scheme (lock-step with a number of parallel exchanges of 1) in the base specification together with performance-enhancing advanced mechanisms.

The present specification is based on the approved text in the [RFC7252] base specification. It is making use of the text that permits advanced congestion control mechanisms and allows them to change protocol parameters, including NSTART and the binary exponential backoff mechanism. Note that Section 4.8 of [RFC7252] limits the leeway that implementations have in changing the CoRE protocol parameters.

The present specification also assumes that, outside of exchanges, non-confirmable messages can only be used at a limited rate without an advanced congestion control mechanism (this is mainly relevant for [RFC7641]). It is also intended to address the [RFC5405] guideline about combining congestion control state for a destination; and to clarify its meaning for CoAP using the definition of an endpoint.

The present specification does not address multicast or dithering beyond basic retransmission dithering.

3. Area of Applicability

The present algorithm is intended to be generally applicable. The objective is to be "better" than default CoAP congestion control in a number of characteristics, including achievable goodput for a given offered load, latency, and recovery from bursts, while providing more predictable stress to the network and the same level of safety from catastrophic congestion. It does require three state variables per scope plus the state needed to do RTT measurements, so it may not be applicable to the most constrained devices (class 1 as per [RFC7228]).

The scope of each instance of the algorithm in the current set of evaluations has been the five-tuple, i.e., CoAP + endpoint (transport address) for Initiator and Responder. Potential applicability to larger scopes needs to be examined.

4. Advanced CoAP Congestion Control: RTO Estimation

For an initiator that plans to make multiple requests to one destination endpoint, it may be worthwhile to make RTT measurements in order to obtain a better RTO estimation than that implied by the default initial timeout of 2 to 3 s. This is based on the usual algorithms for RTO estimation [RFC6298], with appropriately extended default/base values, as proposed in Section 4.2.1. Note that such a mechanism must, during idle periods, decay RTO estimates that are shorter or longer than the basic RTO estimate back to the basic RTO estimate, until fresh measurements become available again, as proposed in Section 4.3.

One important consideration not relevant for TCP is the fact that a CoAP round-trip may include application processing time, which may be hard to predict, and may differ between different resources available at the same endpoint. Also, for communications with networks of constrained devices that apply radio duty cycling, large and variable round-trip times are likely to be observed. Servers will only trigger their early ACKs (with a non-piggybacked response to be sent later) based on the default timers, e.g. after 1 s. A client that has arrived at a RTO estimate shorter than 1 s SHOULD therefore use a larger backoff factor for retransmissions to avoid expending all of its retransmissions in the default interval of 2 to 3 s. A proposal for a mechanism with variable backoff factors is presented in Section 4.2.1.

It may also be worthwhile to do RTT estimates not just based on information measured from a single destination endpoint, but also based on entire hosts (IP addresses) and/or complete prefixes (e.g., maintain an RTT estimate for a whole /64). The exact way this can be

used to reduce the amount of state in an initiator is for further study.

4.1. Blind RTO Estimate

The initial RTO estimate for an endpoint is set to 2 seconds (the initial RTO estimate is used as the initial value for both `E_weak_` and `E_strong_` below).

If only the initial RTO estimate is available, the RTO estimate for each of up to NSTART exchanges started in parallel is set to 2 s times the number of parallel exchanges, e.g. if two exchanges are already running, the initial RTO estimate for an additional exchange is 6 seconds.

4.2. Measured RTO Estimate

The RTO estimator runs two copies of the algorithm defined in [RFC6298], as modified in Section 4.2.1: One copy for exchanges that complete on initial transmissions (the "strong estimator", `E_strong_`), and one copy for exchanges that have run into retransmissions, where only the first two retransmissions are considered (the "weak estimator", `E_weak_`). For the latter, there is some ambiguity whether a response is based on the initial transmission or the retransmissions. For the purposes of the weak estimator, the time from the initial transmission counts. Responses obtained after the third retransmission are not used to update an estimator.

The overall RTO estimate is an exponentially weighted moving average ($\alpha = 0.5$ and 0.25 , respectively) computed of the strong and the weak estimator, which is evolved after each contribution to the weak estimator (1) or to the strong estimator (2), from the estimator that made the most recent contribution:

$$\text{RTO} := 0.25 * \text{E_weak_} + 0.75 * \text{RTO} \quad (1)$$

$$\text{RTO} := 0.5 * \text{E_strong_} + 0.5 * \text{RTO} \quad (2)$$

(Splitting this update into the two cases avoids making the contribution of the weak estimator too big in naturally lossy networks.)

4.2.1. Modifications to the algorithm of RFC 6298

This subsection presents three modifications that must be applied to the algorithm of [RFC6298] as per this document. The first two

recommend new parameter settings. The third one is the variable backoff factor mechanism.

The initial value for each of the two RTO estimators is 2 s.

For the weak estimator, the factor K (the RTT variance multiplier) is set to 1 instead of 4. This is necessary to avoid a strong increase of the RTO in the case that the RTTVAR value is very large, which may be the case if a weak RTT measurement is obtained after one or more retransmissions.

If an RTO estimation is lower than 1 s or higher than 3 s, instead of applying a binary backoff factor in both cases, a variable backoff factor is used. For RTO estimations below 1 s, the RTO for a retransmission is multiplied by 3, while for estimations above 3 s, the RTO is multiplied only by 1.5 (this updated choice of numbers to be verified by more simulations). This helps to avoid that exchanges with small initial RTOs use up all retransmissions in a short interval of time and exchanges with large initial RTOs may not be able to carry out all retransmissions within MAX_TRANSMIT_WAIT (93 s).

The binary exponential backoff is truncated at 32 seconds. Similar to the way retransmissions are handled in the base specification, they are dithered between $1 \times \text{RTO}$ and $\text{ACK_RANDOM_FACTOR} \times \text{RTO}$.

4.2.2. Discussion

In contrast to [RFC6298], this algorithm attempts to make use of ambiguous information from retransmissions. This is motivated by the high non-congestion loss rates expected in constrained node networks, and the need to update the RTO estimators even in the presence of loss. Additional investigation is required to determine whether this is indeed justified.

Some evaluation has been done on earlier versions of this specification [Betzler2013]. A more recent (and more comprehensive) reference is [Betzler2015]. Additional investigation is required.

4.3. Lifetime, Aging

The state of the RTO estimators for an endpoint SHOULD be kept as long as possible. If other state is kept for the endpoint (such as a DTLS connection), it is very strongly RECOMMENDED to keep the RTO state alive at least as long as this other state. It MUST be kept for at least 255 s.

If an estimator has a value that is lower than 1 s, and it is left without further update for 16 times its current value, the RTO estimate is doubled. If an estimator has a value that is higher than 3 s, and it is left without further update for 4 times its current value, the RTO estimate is set to be

$$\text{RTO} := 1 \text{ s} + (0.5 * \text{RTO})$$

(Note that, instead of running a timer, it is possible to implement these RTO aging calculations cumulatively at the time the estimator is used next.)

5. Advanced CoAP Congestion Control: Non-Confirmables

(TO DO: Align this with final consensus on -observe!)

A CoAP endpoint MUST NOT send non-confirmables to another CoAP endpoint at a rate higher than defined by this document. Independent of any congestion control mechanisms, a CoAP endpoint can always send non-confirmables if their rate does not exceed 1 B/s.

Non-confirmables that form part of exchanges are governed by the rules for exchanges.

Non-confirmables outside exchanges (e.g., [RFC7641] notifications sent as non-confirmables) are governed by the following rules:

1. Of any 16 consecutive messages towards this endpoint that aren't responses or acknowledgments, at least 2 of the messages must be confirmable.
2. The confirmable messages must be sent under an RTO estimator, as specified in Section 4.
3. The packet rate of non-confirmable messages cannot exceed $1/\text{RTO}$, where RTO is the overall RTO estimator value at the time the non-confirmable packet is sent.

5.1. Discussion

This is relatively conservative. More advanced versions of this algorithm could run a TFRC-style Loss Event Rate calculator [RFC5348] and apply the TCP equation to achieve a higher rate than $1/\text{RTO}$.

6. Advanced CoAP Congestion Control: Aggregate Congestion Control

(This section is still more experimental than the previous ones.)

6.1. Proposed Algorithm

To avoid possible congestion when sending many packets to different destination endpoints in parallel, the overall number of outstanding interactions towards different destination endpoints should be limited. An upper limit PLIMIT determines the maximum number of outstanding interactions towards different destinations that are allowed in parallel. When a request is sent to a destination endpoint, PLIMIT is determined according to Equation (3) in the case that valid RTO information is already available for the destination endpoint, or using Equation (4) in case that no RTO information is available for the destination endpoint.

$$\text{PLIMIT} = \max(\text{LAMBDA}, \text{LAMBDA} * \text{ACK_TIMEOUT} / \text{mean}(\text{RTO})) \quad (3)$$

$$\text{PLIMIT} = \text{LAMBDA} \quad (4)$$

where LAMBDA determines the minimum value for the maximum number of allowed outstanding interactions and is suggested to be set to 4, and mean(RTO) is the average value of all valid RTO estimations maintained by the device. A new interaction may only be processed if the current overall number of outstanding interactions is lower than the PLIMIT calculated when the request is initiated.

6.2. Example

In the following we give an example, with LAMBDA = 4 (our proposed default LAMBDA):

Assume that a sender has so far obtained RTO estimations for two destination endpoints A (RTO = 0.5 s) and B (RTO = 1.5 s), and currently pcount (a variable which accounts for the number of outstanding interactions towards different endpoints) is equal to 0. Now three transactions are initiated consecutively in the following order: one for A, one for B and one for a new destination C.

When an interaction with node A is initiated, PLIMIT is calculated:

$$\begin{aligned} \text{PLIMIT} &= \max(4, (4 * 2 \text{ s}) / \text{mean}(0.5 \text{ s}, 1.5 \text{ s})) = \max(4, 8 \text{ s} / 1 \text{ s}) = \\ &= \max(4, 8) = 8 \end{aligned}$$

This means that with the current RTO information that the sender has obtained about the destination endpoints, up to 8 outstanding interactions to different endpoints would be allowed. By initiating

an interaction with A, pcount is increased to 1, which is still below PLIMIT. Thus, the interaction may be processed. The same applies to B: pcount increases to 2 after obtaining the same PLIMIT value of 8.

Destination C is unknown to CoCoA, therefore the updated PLIMIT before processing the interaction with node C is 4.

The CoAP request may be processed (pcount = 3). If two more interactions with different unknown destination endpoints would have been initiated, only the first one would have met the requirements to process it (PLIMIT = 4, pcount = 4). The second interaction would have increased pcount to 5, which is not permitted, since PLIMIT is 4. It may occur that pcount exceeds PLIMIT in particular cases, in this case, the interaction is not permitted as well.

6.3. Discussion

The idea of the proposal is to allow more parallel transactions to different destination endpoints if we have low RTO estimations for them (which can be interpreted as good connections and low degree of congestion). If the RTO estimations are large or interactions with unknown destinations are initiated, the mechanism behaves more conservatively by reducing the maximum number of parallel interactions towards different destinations, but allowing at least LAMBDA outstanding interactions. If no RTO information is available for a destination endpoint, PLIMIT is simply set to be LAMBDA.

If at any moment pcount would exceed PLIMIT, CoAP does not immediately perform the transaction. Further, it is important that in parallel, NSTART for each destination endpoint applies (which, for now, we assume to be 1). Overall, LAMBDA determines how aggressive/conservative CoCoA behaves by default and it should be chosen carefully.

It will be necessary to see whether this approach is effective in the sense that it avoids congestion in use cases where transactions to a multitude of different destination endpoints are initiated. An important aspect of such evaluations would be how the choice of LAMBDA affects the performance. On the other hand, a more safe approach would use max(RTO) instead of mean(RTO). Other concerns include the fact that the congestion degree of the paths to "known" endpoints influence whether a new interaction is permitted to some new endpoint which may be in very different conditions in terms of congestion. However, it is desirable to avoid adding a lot of complexity to the current CoCoA mechanisms.

7. IANA Considerations

This document makes no requirements on IANA. (This section to be removed by RFC editor.)

8. Security Considerations

(TBD. The security considerations of, e.g., [RFC5681], [RFC2914], and [RFC5405] apply. Some issues are already discussed in the security considerations of [RFC7252].)

9. Acknowledgements

The first document to examine CoAP congestion control issues in detail was [I-D.eggert-core-congestion-control], to which this draft owes a lot.

Michael Scharf did a review of CoAP congestion control issues that asked a lot of good questions. Several Transport Area representatives made further significant inputs this discussion during IETF84, including Lars Eggert, Michael Scharf, and David Black. Andrew McGregor, Eric Rescorla, Richard Kelsey, Ed Beroaset, Jari Arkko, Zach Shelby, Matthias Kovatsch and many others provided very useful additions.

Authors from Universitat Politecnica de Catalunya have been supported in part by the Spanish Government's Ministerio de Economia y Competitividad through projects TEC2009-11453 and TEC2012-32531, and FEDER.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<http://www.rfc-editor.org/info/rfc2914>>.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, DOI 10.17487/RFC5405, November 2008, <<http://www.rfc-editor.org/info/rfc5405>>.

- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<http://www.rfc-editor.org/info/rfc6298>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

10.2. Informative References

- [Betzler2013] Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "Congestion control in reliable CoAP communication", ACM MSWIM'13 p. 365-372, DOI 10.1145/2507924.2507954, 2013.
- [Betzler2015] Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "CoCoA+: an Advanced Congestion Control Mechanism for CoAP", Ad Hoc Networks Vol. 33 pp. 126-139, DOI 10.1016/j.adhoc.2015.04.007, October 2015.
- [I-D.bormann-core-congestion-control] Bormann, C. and K. Hartke, "Congestion Control Principles for CoAP", draft-bormann-core-congestion-control-02 (work in progress), July 2012.
- [I-D.eggert-core-congestion-control] Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<http://www.rfc-editor.org/info/rfc5348>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://www.rfc-editor.org/info/rfc5681>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015,
<<http://www.rfc-editor.org/info/rfc7641>>.

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

August Betzler
Universitat Politecnica de Catalunya/Fundacio i2CAT
Departament d'Enginyeria Telematica
C/Jordi Girona, 1-3
Barcelona 08034
Spain

Email: august.betzler@entel.upc.edu

Carles Gomez
Universitat Politecnica de Catalunya/Fundacio i2CAT
Escola d'Enginyeria de Telecomunicacio i Aeroespacial
de Castelldefels
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Phone: +34-93-413-7206
Email: carlesgo@entel.upc.edu

Ilker Demirkol
Universitat Politecnica de Catalunya/Fundacio i2CAT
Departament d'Enginyeria Telematica
C/Jordi Girona, 1-3
Barcelona 08034
Spain

Email: ilker.demirkol@entel.upc.edu

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

C. Bormann
Universitaet Bremen TZI
A. Betzler
Fundacio i2CAT
C. Gomez
I. Demirkol
Universitat Politecnica de Catalunya/Fundacio i2CAT
July 08, 2016

CoAP Simple Congestion Control/Advanced
draft-bormann-core-cocoa-04

Abstract

The CoAP protocol needs to be implemented in such a way that it does not cause persistent congestion on the network it uses. The CoRE CoAP specification defines basic behavior that exhibits low risk of congestion with minimal implementation requirements. It also leaves room for combining the base specification with advanced congestion control mechanisms with higher performance.

This specification defines some simple advanced CoRE Congestion Control mechanisms, Simple CoCoA. It is making use of input from simulations and experiments in real networks.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Context	3
3. Area of Applicability	4
4. Advanced CoAP Congestion Control: RTO Estimation	4
4.1. Blind RTO Estimate	5
4.2. Measured RTO Estimate	5
4.2.1. Modifications to the algorithm of RFC 6298	6
4.2.2. Discussion	6
4.3. Lifetime, Aging	7
5. Advanced CoAP Congestion Control: Non-Confirmables	7
5.1. Discussion	8
6. IANA Considerations	8
7. Security Considerations	8
8. References	8
8.1. Normative References	8
8.2. Informative References	9
Appendix A. Advanced CoAP Congestion Control: Aggregate Congestion Control	10
A.1. Proposed Algorithm	10
A.2. Example 1	10
A.3. Example 2	11
A.4. Discussion	12
Acknowledgements	12
Authors' Addresses	13

1. Introduction

(See Abstract.)

Extended rationale for this specification can be found in [I-D.bormann-core-congestion-control] and [I-D.eggert-core-congestion-control], as well as in the minutes of the IETF 84 CoRE WG meetings.

1.1. Terminology

This specification uses terms from [RFC7252]. In addition, it defines the following terminology:

Initiator: The endpoint that sends the message that initiates an exchange. E.g., the party that sends a confirmable message, or a non-confirmable message conveying a request.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

(Note that this document is itself informational, but it is discussing normative statements.)

The term "byte", abbreviated by "B", is used in its now customary sense as a synonym for "octet".

2. Context

In the Vancouver IETF 84 CoRE meeting, a path forward was defined that includes a very simple basic scheme (lock-step with a number of parallel exchanges of 1) in the base specification together with performance-enhancing advanced mechanisms.

The present specification is based on the approved text in the [RFC7252] base specification. It is making use of the text that permits advanced congestion control mechanisms and allows them to change protocol parameters, including NSTART and the binary exponential backoff mechanism. Note that Section 4.8 of [RFC7252] limits the leeway that implementations have in changing the CoRE protocol parameters.

The present specification also assumes that, outside of exchanges, non-confirmable messages can only be used at a limited rate without an advanced congestion control mechanism (this is mainly relevant for [RFC7641]). It is also intended to address the [RFC5405] guideline about combining congestion control state for a destination; and to clarify its meaning for CoAP using the definition of an endpoint.

The present specification does not address multicast or dithering beyond basic retransmission dithering.

3. Area of Applicability

The present algorithm is intended to be generally applicable. The objective is to be "better" than default CoAP congestion control in a number of characteristics, including achievable goodput for a given offered load, latency, and recovery from bursts, while providing more predictable stress to the network and the same level of safety from catastrophic congestion. It does require three state variables per scope plus the state needed to do RTT measurements, so it may not be applicable to the most constrained devices (class 1 as per [RFC7228]).

The scope of each instance of the algorithm in the current set of evaluations has been the five-tuple, i.e., CoAP + endpoint (transport address) for Initiator and Responder. Potential applicability to larger scopes needs to be examined.

Aggregate Congestion Control (Appendix A) is not yet supported by research as well as the other algorithms in this specification. Its use is more interesting on the cloud side, where a single CoAP endpoint may need to talk to thousands of other endpoints and may need to control the burstiness of the resulting aggregate traffic.

4. Advanced CoAP Congestion Control: RTO Estimation

For an initiator that plans to make multiple requests to one destination endpoint, it may be worthwhile to make RTT measurements in order to obtain a better RTO estimation than that implied by the default initial timeout of 2 to 3 s. This is based on the usual algorithms for RTO estimation [RFC6298], with appropriately extended default/base values, as proposed in Section 4.2.1. Note that such a mechanism must, during idle periods, decay RTO estimates that are shorter or longer than the basic RTO estimate back to the basic RTO estimate, until fresh measurements become available again, as proposed in Section 4.3.

One important consideration not relevant for TCP is the fact that a CoAP round-trip may include application processing time, which may be hard to predict, and may differ between different resources available at the same endpoint. Also, for communications with networks of constrained devices that apply radio duty cycling, large and variable round-trip times are likely to be observed. Servers will only trigger their early ACKs (with a non-piggybacked response to be sent later) based on the default timers, e.g. after 1 s. A client that has arrived at a RTO estimate shorter than 1 s SHOULD therefore use a larger backoff factor for retransmissions to avoid expending all of its retransmissions in the default interval of 2 to 3 s. A proposal

for a mechanism with variable backoff factors is presented in Section 4.2.1.

It may also be worthwhile to do RTT estimates not just based on information measured from a single destination endpoint, but also based on entire hosts (IP addresses) and/or complete prefixes (e.g., maintain an RTT estimate for a whole /64). The exact way this can be used to reduce the amount of state in an initiator is for further study.

4.1. Blind RTO Estimate

The initial RTO estimate for an endpoint is set to 2 seconds (the initial RTO estimate is used as the initial value for both `E_weak_` and `E_strong_` below).

If only the initial RTO estimate is available, the RTO estimate for each of up to `NSTART` exchanges started in parallel is set to 2 s times the number of parallel exchanges, e.g. if two exchanges are already running, the initial RTO estimate for an additional exchange is 6 seconds.

4.2. Measured RTO Estimate

The RTO estimator runs two copies of the algorithm defined in [RFC6298], as modified in Section 4.2.1: One copy for exchanges that complete on initial transmissions (the "strong estimator", `E_strong_`), and one copy for exchanges that have run into retransmissions, where only the first two retransmissions are considered (the "weak estimator", `E_weak_`). For the latter, there is some ambiguity whether a response is based on the initial transmission or the retransmissions. For the purposes of the weak estimator, the time from the initial transmission counts. Responses obtained after the third retransmission are not used to update an estimator.

The overall RTO estimate is an exponentially weighted moving average ($\alpha = 0.5$ and 0.25 , respectively) computed of the strong and the weak estimator, which is evolved after each contribution to the weak estimator (1) or to the strong estimator (2), from the estimator that made the most recent contribution:

$$\text{RTO} := 0.25 * \text{E_weak_} + 0.75 * \text{RTO} \quad (1)$$

$$\text{RTO} := 0.5 * \text{E_strong_} + 0.5 * \text{RTO} \quad (2)$$

(Splitting this update into the two cases avoids making the contribution of the weak estimator too big in naturally lossy networks.)

4.2.1. Modifications to the algorithm of RFC 6298

This subsection presents three modifications that must be applied to the algorithm of [RFC6298] as per this document. The first two recommend new parameter settings. The third one is the variable backoff factor mechanism.

The initial value for each of the two RTO estimators is 2 s.

For the weak estimator, the factor K (the RTT variance multiplier) is set to 1 instead of 4. This is necessary to avoid a strong increase of the RTO in the case that the RTTVAR value is very large, which may be the case if a weak RTT measurement is obtained after one or more retransmissions.

If an RTO estimation is lower than 1 s or higher than 3 s, instead of applying a binary backoff factor in both cases, a variable backoff factor is used. For RTO estimations below 1 s, the RTO for a retransmission is multiplied by 3, while for estimations above 3 s, the RTO is multiplied only by 1.5 (this updated choice of numbers to be verified by more simulations). This helps to avoid that exchanges with small initial RTOs use up all retransmissions in a short interval of time and exchanges with large initial RTOs may not be able to carry out all retransmissions within MAX_TRANSMIT_WAIT (93 s).

The binary exponential backoff is truncated at 32 seconds. Similar to the way retransmissions are handled in the base specification, they are dithered between $1 \times \text{RTO}$ and $\text{ACK_RANDOM_FACTOR} \times \text{RTO}$.

4.2.2. Discussion

In contrast to [RFC6298], this algorithm attempts to make use of ambiguous information from retransmissions. This is motivated by the high non-congestion loss rates expected in constrained node networks, and the need to update the RTO estimators even in the presence of loss. Additional investigation is required to determine whether this is indeed justified.

Some evaluation has been done on earlier versions of this specification [Betzler2013]. A more recent (and more comprehensive) reference is [Betzler2015]. Additional investigation is required.

4.3. Lifetime, Aging

The state of the RTO estimators for an endpoint SHOULD be kept as long as possible. If other state is kept for the endpoint (such as a DTLS connection), it is very strongly RECOMMENDED to keep the RTO state alive at least as long as this other state. It MUST be kept for at least 255 s.

If an estimator has a value that is lower than 1 s, and it is left without further update for 16 times its current value, the RTO estimate is doubled. If an estimator has a value that is higher than 3 s, and it is left without further update for 4 times its current value, the RTO estimate is set to be

$$\text{RTO} := 1 \text{ s} + (0.5 * \text{RTO})$$

(Note that, instead of running a timer, it is possible to implement these RTO aging calculations cumulatively at the time the estimator is used next.)

5. Advanced CoAP Congestion Control: Non-Confirmables

A CoAP endpoint MUST NOT send non-confirmables to another CoAP endpoint at a rate higher than defined by this document. Independent of any congestion control mechanisms, a CoAP endpoint can always send non-confirmables if their rate does not exceed 1 B/s.

Non-confirmables that form part of exchanges are governed by the rules for exchanges.

Non-confirmables outside exchanges (e.g., [RFC7641] notifications sent as non-confirmables) are governed by the following rules:

1. Of any 16 consecutive messages towards this endpoint that aren't responses or acknowledgments, at least 2 of the messages must be confirmable.
2. The confirmable messages must be sent under an RTO estimator, as specified in Section 4.
3. The packet rate of non-confirmable messages cannot exceed $1/\text{RTO}$, where RTO is the overall RTO estimator value at the time the non-confirmable packet is sent.

5.1. Discussion

This is relatively conservative. More advanced versions of this algorithm could run a TFRC-style Loss Event Rate calculator [RFC5348] and apply the TCP equation to achieve a higher rate than $1/\text{RTO}$.

[RFC7641], Section 4.5.1, specifies that the rate of NONs SHOULD NOT exceed $1/\text{RTT}$ on average, if the server can maintain an RTT estimate for a client. CoCoA limits the packet rate of NONs in this situation to $1/\text{RTO}$. Assuming that the RTO estimation in CoCoA works as expected, $\text{RTO}[k]$ should be slightly greater than the $\text{RTT}[k]$, thus CoCoA would be more conservative. The expectation therefore is that complying with the NON rate set by CoCoA leads to complying with [RFC7641].

6. IANA Considerations

This document makes no requirements on IANA. (This section to be removed by RFC editor.)

7. Security Considerations

(TBD. The security considerations of, e.g., [RFC5681], [RFC2914], and [RFC5405] apply. Some issues are already discussed in the security considerations of [RFC7252].)

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<http://www.rfc-editor.org/info/rfc2914>>.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, DOI 10.17487/RFC5405, November 2008, <<http://www.rfc-editor.org/info/rfc5405>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<http://www.rfc-editor.org/info/rfc6298>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

8.2. Informative References

- [Betzler2013] Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "Congestion control in reliable CoAP communication", ACM MSWIM'13 p. 365-372, DOI 10.1145/2507924.2507954, 2013.
- [Betzler2015] Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "CoCoA+: an Advanced Congestion Control Mechanism for CoAP", Ad Hoc Networks Vol. 33 pp. 126-139, DOI 10.1016/j.adhoc.2015.04.007, October 2015.
- [I-D.bormann-core-congestion-control] Bormann, C. and K. Hartke, "Congestion Control Principles for CoAP", draft-bormann-core-congestion-control-02 (work in progress), July 2012.
- [I-D.eggert-core-congestion-control] Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<http://www.rfc-editor.org/info/rfc5348>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://www.rfc-editor.org/info/rfc5681>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

Appendix A. Advanced CoAP Congestion Control: Aggregate Congestion Control

(The mechanism defined in this appendix has received less research than the ones in the main body of this specification.)

A.1. Proposed Algorithm

To avoid possible congestion when sending many packets to different destination endpoints in parallel, the overall number of outstanding interactions towards different destination endpoints should be limited. An upper limit PLIMIT determines the maximum number of outstanding interactions towards different destination endpoints that are allowed in parallel. When a request is to be sent to a destination endpoint, PLIMIT is determined according to Equation (3) in the case that no RTO information is already available for the destination endpoint, or using Equation (4) in case that valid RTO information is available for the destination endpoint. Both formulas use LAMBDA, as defined in Equation (5).

$$\text{PLIMIT} = \text{LAMBDA} \quad (3)$$

$$\text{PLIMIT} = \max(\text{LAMBDA}, \text{LAMBDA} * \text{ACK_TIMEOUT} / \text{mean}(\text{RTO})) \quad (4)$$

$$\text{LAMBDA} = \max(4, \text{KNOWN_DEST_ENDPOINTS} / 4) \quad (5)$$

mean(RTO) is the average value of all valid RTO estimations maintained by the device. LAMBDA is the maximum of a constant value (4 by default) and the rounded up value of KNOWN_DEST_ENDPOINTS/4, where KNOWN_DEST_ENDPOINTS is the overall number of "known" destination endpoints (i.e. destination endpoints for which an RTO estimate is maintained).

A new interaction may only be processed if the current overall number of outstanding interactions is lower than the PLIMIT calculated when the request is initiated.

A.2. Example 1

In the following we give an example, with LAMBDA = 4 (our proposed default LAMBDA):

Assume that a sender has so far obtained RTO estimations for two destination endpoints A (RTO = 0.5 s) and B (RTO = 1.5 s), and currently pcount (a variable which accounts for the number of outstanding interactions towards endpoints) is equal to 0. Now three transactions are initiated consecutively in the following order: one for A, one for B and one for a new destination C.

When an interaction with node A is initiated, LAMBDA is calculated

$$\text{LAMBDA} = \max(4, 3/4) = 4.$$

Then PLIMIT is calculated:

$$\text{PLIMIT} = \max(4, (4*2 \text{ s})/\text{mean}(0.5 \text{ s}, 1.5 \text{ s})) = \max(4, 8 \text{ s}/1 \text{ s}) = \max(4, 8) = 8$$

This means that with the current RTO information the sender has obtained about the destination endpoints, up to 8 outstanding interactions to different destination endpoints would be allowed. By initiating an interaction with A, pcount is increased to 1, which is still below PLIMIT. Thus, the interaction may be processed. The same applies to B: pcount increases to 2 after obtaining the same PLIMIT value of 8.

Destination C is unknown to CoCoA, therefore the updated PLIMIT before processing the interaction with node C is 4. The CoAP request may be processed (pcount = 3). If two more interactions with different unknown destination endpoints would have been initiated, only the first one would have met the requirements to process it (PLIMIT = 4, pcount = 4). The second interaction would have increased pcount to 5, which is not permitted, since PLIMIT is 4. It may occur that pcount exceeds PLIMIT in particular cases, in this case, the interaction is not permitted as well. If the number of destinations exchanges are initiated with would increase further, eventually LAMBDA could grow beyond 4, allowing for more interactions to be sent in parallel.

A.3. Example 2

Let us now assume that a sender has so far obtained RTO estimations for 101 destination endpoints, their average RTO is 1 s, and currently pcount is equal to 0. When a new transaction is initiated with a destination endpoint for which an RTO estimate is available, LAMBDA is calculated

$$\text{LAMBDA} = \max(4, 101/4) = 26$$

Based on this, PLIMIT is calculated as follows:

$$\text{PLIMIT} = \max(26, (26*2 \text{ s})/1 \text{ s}) = \max(26, 52) = 52$$

This means that with the current RTO information that the sender has obtained about the destination endpoints, up to 52 outstanding interactions to known destination endpoints would be allowed.

However, if the new exchange is to be initiated with an "unknown" destination endpoint (i.e. an endpoint for which an RTO estimate is not available), then PLIMIT would be 26.

A.4. Discussion

The idea of the proposal is to allow more parallel transactions to different destination endpoints if we have low RTO estimations for them (which can be interpreted as good connections and low degree of congestion). If the RTO estimations are large or interactions with unknown destinations are initiated, the mechanism behaves more conservatively by reducing the maximum number of parallel interactions towards different destinations, but allowing at least LAMBDA outstanding interactions. The second term of the `max()` statement used to calculate LAMBDA avoids behaving too restrictively when exchanges with many different destination endpoints are initiated. If no RTO information is available for a destination endpoint, PLIMIT is simply set to be LAMBDA.

If at any moment `pcount` would exceed PLIMIT, CoAP does not immediately perform the transaction. Further, it is important that in parallel, NSTART for each destination endpoint applies (which, for now, we assume to be 1). The default value used for LAMBDA (equal to 4 as per this document) determines how aggressive/conservative CoCoA behaves by default for a limited set of destination endpoints and it should be chosen carefully. The term `KNOWN_DEST_ENDPOINTS/4` loosens the hard limit of exchanges when large numbers of destination endpoints are addressed.

It will be necessary to see whether this approach is effective in the sense that it avoids congestion in use cases where transactions to a multitude of different destination endpoints are initiated. An important aspect of such evaluations would be whether LAMBDA is too conservative when dealing with few destination endpoints and whether it allows for a dynamic adjustment of parallel exchanges with large numbers of destination endpoints. On the other hand, a more safe approach would use `max(RTO)` instead of `mean(RTO)`. Other concerns include the fact that the congestion degree of the paths to "known" destination endpoints influence whether a new interaction is permitted to some new endpoint which may be in very different conditions in terms of congestion. However, it is desirable to avoid adding a lot of complexity to the current CoCoA mechanisms.

Acknowledgements

The first document to examine CoAP congestion control issues in detail was [I-D.eggert-core-congestion-control], to which this draft owes a lot.

Michael Scharf did a review of CoAP congestion control issues that asked a lot of good questions. Several Transport Area representatives made further significant inputs this discussion during IETF84, including Lars Eggert, Michael Scharf, and David Black. Andrew McGregor, Eric Rescorla, Richard Kelsey, Ed Beroaset, Jari Arkko, Zach Shelby, Matthias Kovatsch and many others provided very useful additions.

Authors from Universitat Politecnica de Catalunya have been supported in part by the Spanish Government's Ministerio de Economia y Competitividad through projects TEC2009-11453 and TEC2012-32531, and FEDER.

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336. His contribution to this work has been carried out in part during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge, in collaboration with Prof. Jon Crowcroft.

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

August Betzler
Fundacio i2CAT
Mobile and Wireless Internet Group
C/ del Gran Capita, 2
Barcelona 08034
Spain

Email: august.betzler@entel.upc.edu

Carles Gomez
Universitat Politecnica de Catalunya/Fundacio i2CAT
Escola d'Enginyeria de Telecomunicacio i Aeroespacial
de Castelldefels
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Phone: +34-93-413-7206
Email: carlesgo@entel.upc.edu

Ilker Demirkol
Universitat Politecnica de Catalunya/Fundacio i2CAT
Departament d'Enginyeria Telematica
C/Jordi Girona, 1-3
Barcelona 08034
Spain

Email: ilker.demirkol@entel.upc.edu

INTERNET-DRAFT
Intended Status: Standards Track
Expires: September 22, 2016

Luyuan Fang
Microsoft

March 21, 2016

Failure Detection Extensions for Publish-Subscribe in CoAP
draft-fang-core-coap-pubsub-failure-detection-00

Abstract

This document defines extensions to the Constrained Application Protocol Publish/Subscribe function set, to make the protocol suitable to address the use case of failure detection in a hyper-scale system with millions of endpoints. Specifically, this document defines a Last Will mechanism and a scheme to guarantee hot fail-over of the pub/sub broker.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. Pub/Sub Broker for CoAP	5
3. Last Will and Testament	6
4. Pub/Sub Broker Fail-Over	7
5. Security Considerations	7
6. IANA Considerations	7
7. References	7
7.1 Normative References	7
7.2 Informative References	7
Authors' Addresses	8

1. Introduction

Many new protocols are being specified, and many existing ones are evolving, to meet the scalability, functionality, and footprint requirements of the Internet of Things (IoT), or Web of Things (WoT).

These protocols include Constrained Application Protocol (CoAP) [RFC7252], the Message Queuing Telemetry Transport (MQTT) protocol [MQTT], the Advanced Message Queuing Protocol (AMQP) [AMQP], and the Streaming Text Oriented Messaging Protocol (STOMP) [STOMP], among others. The Extensible Messaging and Presence Protocol (XMPP) [RFC7622] and HTTP/2 [RFC7540] also provide many capabilities that make them very suitable to support IoT use cases.

Although the proliferation of protocols for use in IoT is a clear indication that there is no single "silver bullet" protocol that can optimally address all the emerging IoT use cases, all these protocols are generally designed to provide connectivity to massive numbers of rather simple devices, typically resource constrained in terms of computing power, battery life, bandwidth, and reachability. As such, the design emphasis in these protocols is on scalability, small footprint, efficient use of available bandwidth, ease of parsing and processing, and client independency. To achieve these design objectives, these protocols have introduced several interesting and useful concepts to remove limitations in existing protocol and provide effective solutions to the new requirements.

As these protocols continue to mature, extensions are specified to increase the scope of their use (and, arguably, perhaps have one protocol prevail over others in a sort of "war of protocols" that is ensuing). In these extensions, it is often the case that a protocol is augmented with some desired characteristics or concepts already demonstrated by other protocols. For example, MQTT for Sensor Networks (MQTT-SN) [MQTTSN] is a flavor of MQTT that substitutes TCP with UDP, to achieve better scalability and lower complexity in certain use cases. Directly relevant to this document, recent work in IETF [I-D.draft-koster-core-coap-pubsub] is meant to add the desirable Publish/Subscribe (pub/sub) message paradigm, which distinguishes MQTT, AMQP, and other protocols, to CoAP.

Because of their desirable characteristics, the usefulness of these protocols is not necessarily confined to IoT use cases, but these protocol become strong candidates to address any use case where scalability, simplicity, and responsiveness are paramount. One of such use cases is fault detection in a hyper-scale network with millions or tens of millions of endpoints, such as a Data Center (DC). In a DC, many fault detection, diagnostics, and fault recovery mechanisms are typically deployed. However, as the scale and

complexity of the DC increases, there is an emerging need to devise new light-weight, scalable, device-agnostic, massively distributable, reactive mechanisms to assist and complement existing ones.

The simplicity, efficiency, and scalability of CoAP makes it a frontrunner as an interesting solution for the fault-detection use case. The addition of the pub/sub paradigm and the corresponding introduction of a pub/sub broker for CoAP

[I-D.draft-koster-core-coap-pubsub] further provide a convenient, scalable architecture for fault detection, where the broker can rapidly detect the occurrence of faults in the connected clients (e.g., nodes in the DC) and propagate the information to interested listener in a timely fashion.

CoAP with pub/sub mechanism is an important ingredient for solving the use case, but of course it is not the only one. This document further extends the CoAP pub/sub function set with two additional, useful mechanisms for this purpose, thus making CoAP an even stronger candidate as a lightweight protocol solution for fault detection.

First, this document specifies a Last Will and Testament (LWT) mechanism to be added to the CoAP pub/sub function set. The LWT mechanism, which is used in other protocols such as MQTT, is explicitly designed to define the behavior of the broker in case of unexpected loss of connectivity with a client, as it is indeed the case when a fault occurs. The LWT mechanism is most effective when it is used in conjunction with some sort of Keep Alive mechanism, which should also be defined as part of the specification.

A well-known shortcoming of the pub/sub paradigm is the fact that the broker becomes a single point of failure. Clearly, this problem is extremely relevant in the use case at hand, where the broker is a key component of the fault detection architecture. This document further defines extensions to support redundancy among brokers, and achieve hot fail-over in case of failure of the brokers themselves.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document uses terms and concepts that are discussed in [RFC5988], [RFC6690], [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format [RFC6570] is also used in this specification.

This specification makes use of the following additional terminology,

defined in [I-D.draft-koster-core-coap-pubsub]:

- o Publish-Subscribe (pub/sub): A messaging paradigm where a publisher publishes messages to a broker and interested receivers subscribe to the broker to receive messages. The published messages are delivered by the broker to the subscribed receivers.
- o CoAP pub/sub function set: A group of REST resources that together provide the CoAP pub/sub service.
- o CoAP pub/sub Broker: A server node capable of receiving messages from publishers and sending messages to subscribed receivers.
- o CoAP pub/sub Client: A CoAP client that implements the CoAP pub/sub function set.
- o Topic: A unique identifier for a particular item being published and/or subscribed to. The broker uses the topics to match subscriptions with publications.
- o CoAP pub/sub Function Set: The interface between a CoAP pub/sub Broker and pub/sub Clients.

In addition, this document uses the following terms.

Term	Definition
-----	-----
AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol
CSP	Cloud Service Provider
DC	Data Center
IoT	Internet of Things
LWT	Last Will and Testament
MQTT	Message Queuing Telemetry Transport
MQTT-SN	MQTT for Sensor Networks
SDN	Software Defined Network
STOMP	Constrained Application Protocol
SVR	Server
WoT	Web of Things
XMPP	Extensible Messaging and Presence Protocol

2. Pub/Sub Broker for CoAP

The Pub/Sub Broker architecture and pub/sub function set is defined in [I-D.draft-koster-core-coap-pubsub].

The CoAP pub/sub Broker is a CoAP Server that exposes an interface

for CoAP clients to perform publish/subscribe interactions. The Broker typically has resource to buffer messages that are published by the CoAP clients. The Broker matches a published resource/message with the interested listener using Topics. Listeners subscribe to specific topics to receive information published by specific clients.

The CoAP pub/sub function set as defined in [I-D.draft-koster-core-coap-pubsub] provides the following operations.

- o DISCOVER. Used by CoAP clients to discover CoAP pub/sub Brokers
- o CREATE. Used by CoAP clients to create a topic.
- o PUBLISH. Used by CoAP clients to update a specific topic on the broker (i.e., publish a message on a topic).
- o SUBSCRIBE. Used by CoAP clients (listeners) to subscribe to topics.
- o UNSUBSCRIBE. Used by CoAP clients (listeners) to unsubscribe to topics.
- o READ. Used by a CoAP client (listener) to obtain the most recent published value on a topic. Useful when a client first joins or re-joins the pub/sub system.
- o REMOVE. Used by a CoAP client to remove an existing topic.

3. Last Will and Testament

The Last Will and Testament (LWT) mechanism is used to define the behavior of the broker in case of unexpected loss of connectivity with a client. This may be the result of an error detected by the broker, or may be triggered by the client failing to communicate with the broker within a Keep Alive.

In order to implement the LWT mechanism, the CoAP pub/sub function set needs to be extended by adding:

- i. a mechanism to create a WILL topic;
- ii. a mechanism to specify a WILL message.

The WILL message is the message that the broker MUST post on the WILL topic in case a failure of the corresponding CoAP client is detected.

These two mechanisms are implemented by modifying the CREATE

operation in the CoAP pub/sub function set.

4. Pub/Sub Broker Fail-Over

TBD.

5. Security Considerations

TBD.

6. IANA Considerations

TBD.

7. References

7.1 Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<http://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7622] P. Saint-Andre et al., "Extensible Messaging and Presence Protocol (XMPP): Address Format", RFC 6122, September 2015, <<https://tools.ietf.org/html/rfc7622>>.
- [RFC7540] M. Belshe et al., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, May 2015, <<https://tools.ietf.org/html/rfc7540>>.

7.2 Informative References

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc->

editor.org/info/rfc5988>.

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-05 (work in progress), October 2015.

[I-D.draft-koster-core-coap-pubsub] M. Koster et al., "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-04 (work in progress), November 2015.

[AMQP] "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0", OASIS Standard, October 2012, <<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html>>.

[MQTT] "MQTT Version 3.1.1", OASIS Standard, October 2014, <<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>>.

[MQTTSN] "MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2", November 2013, <http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf>.

[STOMP] "STOMP Protocol Specification, Version 1.2", <<https://stomp.github.io/stomp-specification-1.2.html>>.

Authors' Addresses

Luyuan Fang
Microsoft
15590 NE 31st St
Redmond, WA 98052
Email: lufang@microsoft.com

CORE
Internet-Draft
Intended status: Standards Track
Expires: September 1, 2016

T. Fossati
Nokia
H. Tschofenig
ARM Ltd.
February 29, 2016

Introducing Server Name Identifiers in Certificates
draft-fossati-core-server-name-id-00.txt

Abstract

TBD.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 1, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology and Requirements Language	3
3. Syntax	3
4. Server Name Indication (SNI) Name Type and Server Name Syntax	4
5. Subject Alternative Name Extension	4
6. Client Behaviour	5
7. Server Behaviour	5
8. Example	6
9. IANA Considerations	7
10. Security Considerations	8
11. Acknowledgements	8
12. References	8
12.1. Normative References	8
12.2. Informative References	9
Authors' Addresses	9

1. Introduction

Today, many Internet of Things (IoT) deployments consist of an IoT device that interacts with a cloud service infrastructure. (This deployment model is described in Section 2.2 of [RFC7452].) If TLS/DTLS is used to mutually authenticate the device and the cloud server, then the guidance in [I-D.ietf-dice-profile] – which, in turn, takes [RFC7252] recommendations into account – should be followed.

Let us take the CoAP protocol as an example. According to Section 9.1.3.3 of [RFC7252], a DTLS client that receives a certificate from the DTLS server must check that the authority of the requested URI matches "at least one of the authorities of any CoAP URI found in a field of URI type in the SubjectAltName (SAN) set. If there is no SubjectAltName in the certificate, then the authority of the request URI must match the Common Name (CN) found in the certificate [...].". A URI that includes an authority, such as a 'coaps' URI, needs to include a fully qualified domain name (FQDN), or an IP literal as its host part, as stated in Section 4.2.1.6 of [RFC5280]. So, an IoT device that wants to talk to a CoAP server at coaps://example.com will expect to receive a certificate with a matching URI in either the content of the SAN extension or the CN.

The Server Name Indication (SNI) extension [RFC6066] defined for TLS/DTLS allows a client to tell a server the name of the server it is contacting. This is a feature useful when the server is part of a hosting solution where multiple virtual servers are using a single underlying network address. Section 3 of [RFC6066] only allows FQDN hostname of the server in the ServerName field.

When a TLS/DTLS server has an FQDN registered in the DNS then the use of certificates work well with TLS/DTLS to secure protocols like HTTP or CoAP. While the DNS can be taken for granted in the Web, many IoT deployments do not mandate its presence. There are even IoT deployments where the server infrastructure is located in a residential environment in which IoT devices interact with the server solely in the local network (see also Section 2.1 of [RFC7452]).

Since static configuration is not generally a viable option from a usability point of view, in order to cope with scenarios like the one described above there is a need to define some kind of stable, non-DNS-based identifier that can be used with certificates. Note that an alternative is to avoid using certificates altogether and to instead use raw public keys. With raw public keys, the raw public key itself is the identifier and some out-of-band validation technique is needed instead, as described in RFC 7250 [RFC7250].

This document specifies such identifiers for use with certificates.

2. Terminology and Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Syntax

[Editor's Note: This is a strawman proposal for the identifier definition.]

This section defines the syntax for the instance and the domain component, which are separated by the "@" sign. The structure for the name and the domain component are determined by the namespace prefix. We call this new construct the 'Server Name Identifier' (SN-ID).

The following ABNF reuses ALPHA and DIGIT from [RFC5234].

```
char = ALPHA / DIGIT / "-" / "_" / "~" / "!" /  
      "$" / "&" / "'" / "(" / ")" / "*" /  
      "," / ";" / "="  
ns = ALPHA *(ALPHA / DIGIT / "-")  
name = 1*RD-char  
domain = 1*63RD-char  
authority = [ ns "." ] name [ "@" domain ]
```

Note that the "@" and the "." signs are illegal characters in the name and the ns components.

Here are some examples:

- o eui64.01-23-45-67-89-ab-cd-ef
- o imei.+123456789012345
- o uuid.84c05e54-1d3c-48b6-bf12-c11c55f8fdac@foo

4. Server Name Indication (SNI) Name Type and Server Name Syntax

In order to encode the SN-ID in a `ServerNameList`, the `extension_data` field of the `server_name` extension is expanded to allow the SN-ID in a `ServerName` extension:

```
struct {
    NameType name_type;
    select (name_type) {
        case host_name: HostName;
        case sn_id: AuthorityType;
    } name;
} ServerName;

enum {
    host_name(0),
    sn_id(1),
    (255)
} NameType;

opaque AuthorityType<1..2^16-1>;
```

`AuthorityType`, the data structure associated with the authority declaration, is a variable-length vector that begins with a 16-bit length field indicating the length of the following authority. The value in the authority field is represented as a byte string using ASCII encoding. It MUST NOT contain any percent-encoded character other than for those characters not explicitly allowed by the grammar in Section 3.

5. Subject Alternative Name Extension

As described in RFC 5280 [RFC5280] the `subjectAltName` may carry additional name types through the use of the `otherName` field. The format and semantics of the name are indicated through the `OBJECT IDENTIFIER` in the `type-id` field. The name itself is conveyed as value field in `otherName`. This document defines a new value for the `type-id` field.

This section defines the SN-ID as a form of otherName from the GeneralName structure in SubjectAltName defined in [RFC5280].

id-sn-id OBJECT IDENTIFIER ::= { id-pkix id-sn-id(TODO) }

An X.509 server certificate intended to be used with this specification MUST contain an otherName SAN identified using a type-id of 'id-sn-id-san'.

id-sn-id-san OBJECT IDENTIFIER ::= { id-sn-id 2 }

The value field of the otherName MUST contain the SN-ID, as described in Section 3, encoded as a IA5String.

6. Client Behaviour

TLS/DTLS clients behave as follows:

- 1) Clients MAY include an extension of type "server_name" in the (extended) client hello. A client supporting this specification MAY include one (and one only) ServerName element conveying the SN-ID.
- 2) Process the Certificate message, verify the digital signature and perform path validation (as described in Section 3.2 of RFC 5280).
- 3) Verify that the intended server name is indeed one of the identities bound to the presented certificate, by checking that the name in the SAN otherName of type id-sn-id-san matches the authority requested via server_name.
- 4) Upon receiving the CertificateRequest message, send the certificate via a Certificate message - or CertificateURL message, if the client_certificate_url extension has been successfully negotiated during the "hello" exchange.
- 5) Send ClientKeyExchange and then CertificateVerify to complete the mutual authentication process.

7. Server Behaviour

TLS/DTLS servers behave as follows:

- 1) A server receiving the extended ClientHello carrying a server_name extension uses the given server_name (with the included SN-ID) to select the appropriate certificate. The selected certificate MUST include a SAN otherName with an id-sn-

id-san type-id and value, which MUST match the requested ServerName;

- a) If no certificate can be selected, the server MUST terminate the handshake by sending a fatal-level unrecognized_name(112) alert. [[CREF1: Prefer a single, hard failure, path over soft failure, or worse: ignoring the error altogether. Rationale: do not waste time/energy; provide clear and prompt diagnostic to the peer. It doesn't look like the condition that could be exploited by a timing attack.]]
 - b) If a matching certificate exist, the server SHALL include an extension of type "server_name" in the (extended) ServerHello message with an empty value.
- 2) The server MUST send the selected certificate to the client in the Certificate message.
 - 3) Server MAY request a client certificate via a CertificateRequest message and conclude its negotiation with a ServerHelloDone message.
 - 4) When server receives the Certificate message from the client it MUST process the Certificate message, verify the digital signature of the certificate and perform path validation (as described in Section 3.2 of RFC 5280).
 - a) If the client certificate processing fails then the server MUST tear down the exchange.
 - b) If the client certificate processing is successful then the server finalizes the TLS handshake.
 - 5) The server application running on top of the TLS/DTLS stack MUST check the included client identity against the access control policy at the server. It is important to note that this verification check is done outside the TLS/DTLS stack; failure to do it at the application layer may result in unauthorized access.

8. Example

In this section we discuss a more complete scenario where the mechanism described in this document is practical. Consider the following setup where IoT devices are located in a small home network with a Resource Directory (RD) [I-D.ietf-core-resource-directory] helping with discovery.

A resource directory is an entity that acts as a centralized store where protocol endpoints can register their resources and thereby make them available to others. Other devices subsequently use the resource directory to lookup devices and their resources.

The RD defines the concept of an "endpoint name" which identifies a given endpoint within a "domain". Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of [I-D.ietf-core-resource-directory] an endpoint is used to describe a web server that registers resources to the resource directory. An endpoint is identified by its endpoint name, which is included during registration, and is unique within the associated domain of the registration.

Imagine various IoT devices registering their resources with the pre-configured RD (or dynamically discovered RD). Section 5.2 of [I-D.ietf-core-resource-directory] contains a description of registration procedure using CoAP and offers examples. The resource server stores, among other things, the endpoint name and (optionally) a domain.

Once the resources are registered nodes may use the resource directory to discover the resources offered by others. Section 7 of [I-D.ietf-core-resource-directory] describes the discovery procedure and lists examples. A node may, for example, search for resources of type 'temperature' and learns the network addresses of the nodes hosting those resources as well as their endpoint name (and, if available, their domain).

Once the network address has been obtained, direct communication between the two entities can be initiated. During the subsequent DTLS exchange to secure CoAP the server hosting the resources offers his certificates and the client executes the steps outlined in Section 6 to match the endpoint name (and optionally the domain) learned through the resource directory with the SN-ID provided in the server certificate. Note that it is not envisioned that the client compares the input to the discovery procedure with the SN-ID. In this example the input to the discovery procedure with the resource directory was the resource type, i.e., the 'temperature' string. It is therefore assumed that the client trusts the resource directory to return genuine mappings from abstract search terms to specific servers hosting those resources.

9. IANA Considerations

TBD: This document requires registration of various identifiers into existing registries, namely

- o id-sn-id
- o OtherName.type-id::id-sn-id-san
- o NameType::sn-id
- o ServerName.name::Authority

10. Security Considerations

It's the responsibility of the CA issuing the certificate to verify the content of the certificate before issuing a new certificate. In particular, the CA MUST ensure uniqueness of the issued certificates and that the included SN-ID is indeed correct. This should exclude the threat of a (possibly rogue) node to successfully impersonate another node's identity.

Security considerations from Section 11.1 of [RFC6066] fully apply.

11. Acknowledgements

We would like to thank Martin Thomson, Carsten Bormann, Andrew McGregor, and Zach Shelby for their feedback during IETF 92.

12. References

12.1. Normative References

- [I-D.ietf-core-resource-directory]
Shelby, Z. and C. Bormann, "CoRE Resource Directory",
draft-ietf-core-resource-directory-02 (work in progress),
November 2014.
- [I-D.ietf-dice-profile]
Tschafenig, H. and T. Fossati, "TLS/DTLS Profiles for the
Internet of Things", draft-ietf-dice-profile-17 (work in
progress), October 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax
Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/
RFC5234, January 2008,
<<http://www.rfc-editor.org/info/rfc5234>>.

- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<http://www.rfc-editor.org/info/rfc5280>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<http://www.rfc-editor.org/info/rfc6066>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

12.2. Informative References

- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<http://www.rfc-editor.org/info/rfc7250>>.
- [RFC7452] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC 7452, DOI 10.17487/RFC7452, March 2015, <<http://www.rfc-editor.org/info/rfc7452>>.

Authors' Addresses

Thomas Fossati
Nokia
3 Ely Road
Milton, Cambridge CB24 6DD
Great Britain

Email: thomas.fossati@nokia.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
Great Britain

Email: Hannes.tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

Thing-to-Thing Research Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2019

K. Hartke
Ericsson
October 22, 2018

CoRE Applications
draft-hartke-core-apps-08

Abstract

The application programmable interfaces of RESTful, hypermedia-driven Web applications consist of a number of reusable components such as Internet media types and link relation types. This document proposes "CoRE Applications", a convention for application designers to build the interfaces of their applications in a structured way, so that implementers can easily build interoperable clients and servers, and other designers can reuse the components in their own applications.

Note to Readers

This Internet-Draft should be discussed on the Thing-to-Thing Research Group (T2TRG) mailing list <t2trg@irtf.org> <<https://www.irtf.org/mailman/listinfo/t2trg>>.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. CoRE Applications	3
2.1. Communication Protocols	4
2.1.1. URI Schemes	4
2.2. Representation Formats	5
2.2.1. Internet Media Types	5
2.3. Links	7
2.3.1. Link Relation Types	8
2.3.2. Template Variable Names	8
2.4. Forms	8
2.4.1. Form Relation Types	9
2.4.2. Form Field Names	9
2.5. Well-Known Locations	10
3. CoRE Application Descriptions	10
3.1. Template	11
4. URI Design Considerations	12
5. Security Considerations	14
6. IANA Considerations	14
7. References	14
7.1. Normative References	14
7.2. Informative References	15
Acknowledgements	16
Author's Address	17

1. Introduction

Representational State Transfer (REST) [16] is an architectural style for distributed hypermedia systems. Over the years, REST has gained popularity not only as an approach for large-scale information dissemination, but also as the basic principle for designing and building Internet-based applications in general.

In the coming years, the size and scope of the Internet is expected to increase greatly as physical-world objects become smart enough to communicate over the Internet -- a phenomenon known as the Internet of Things (IoT). As things learn to speak the languages of the net,

the idea of applying REST principles to the design of IoT application architectures suggests itself. To this end, the Constrained Application Protocol (CoAP) [23] was created, an application-layer protocol that enables RESTful applications in constrained-node networks [10], giving rise to a new setting for Internet-based applications: the Constrained RESTful Environment (CoRE).

To realize the full benefits and advantages of the REST architectural style, a set of constraints needs to be maintained when designing applications and their application programming interfaces (APIs). One of the fundamental principles of REST is that "REST APIs must be hypertext-driven" [17]. However, this principle is often ignored by application designers. Instead, APIs are specified out-of-band in terms of fixed URI patterns (e.g., in the API documentation or in a machine-readable format that facilitates code generation). Although this approach may appear easy for clients to use, the fixed resource names and data formats lead to a tight coupling between client and server implementations and make the system less flexible [5]. Violations of REST design principles like this result in APIs that may not be as scalable, extensible, and interoperable as promised by REST.

REST is intended for network-based applications that are long-lived and span multiple organizations [17]. Principled REST APIs require some design effort, since application designers do not only have to take current requirements into consideration, but also have to anticipate changes that may be required in the future -- years or even decades after the application has been deployed for the first time. The reward is long-term stability and evolvability, both of which are very desirable features in the Internet of Things.

To aid application designers in the design process, this document proposes "CoRE Applications", a convention for building the APIs of RESTful, hypermedia-driven Web applications. The goal is to help application designers avoid common mistakes by focusing almost all of the descriptive effort on defining the Internet media type(s) that are used for representing resources and driving application state.

A template for a "CoRE Application Description" provides a consistent format for the description of APIs so that implementers can easily build interoperable clients and servers, and other application designers can reuse the components in their own applications.

2. CoRE Applications

A CoRE Application API is a named set of reusable components. It describes a contract between a server hosting an instance of the

described application and clients that wish to interface with that instance.

The API is generally comprised of:

- o communication protocols, identified by URI schemes,
- o representation formats, identified by Internet media types,
- o link relation types,
- o form relation types,
- o template variables in templated links,
- o form field names in forms, and
- o well-known locations.

Together, these components provide the specific, in-band instructions to a client for interfacing with a given application.

2.1. Communication Protocols

The foundation of a hypermedia-driven REST API are the communication protocol(s) spoken between a client and a server. Although HTTP/1.1 [14] is by far the most common communication protocol for REST APIs, a REST API should typically not be dependent on any specific communication protocol.

2.1.1. URI Schemes

The usage of a particular protocol by a client is guided by URI schemes [7]. URI schemes specify the syntax and semantics of URI references [1] that the server includes in hypermedia controls such as links and forms.

A URI scheme refers to a family of protocols, typically distinguished by a version number. For example, the "http" URI scheme refers to the two members of the HTTP family of protocols: HTTP/1.1 [14] and HTTP/2 [8] (as well as some predecessors). The specific HTTP version used is negotiated between a client and a server in-band using the version indicator in the HTTP request-line or the TLS Application-Layer Protocol Negotiation (ALPN) extension [18].

IANA maintains a list of registered URI schemes at
<<http://www.iana.org/assignments/uri-schemes>>.

2.2. Representation Formats

In RESTful applications, clients and servers exchange representations that capture the current or intended state of a resource and that are labeled with a media type. A representation is a sequence of bytes whose structure and semantics are specified by a representation format: a set of rules for encoding information.

Representation formats should generally allow clients with different goals, so they can do different things with the same data. The specification of a representation format "describes a problem space, not a prescribed relationship between client and server. Client and server must share an understanding of the representations they're passing back and forth, but they don't need to have the same idea of what the problem is that needs to be solved." [21]

Representation formats and their specifications frequently evolve over time. It is part of the responsibility of the designer of a new version to insure both forward and backward compatibility: new representations should work reasonably (with some fallback) with old processors and old representations should work reasonably with new processors [20].

Representation formats enable hypermedia-driven applications when they support the expression of hypermedia controls such as links (Section 2.3) and forms (Section 2.4).

2.2.1. Internet Media Types

One of the most important aspect of hypermedia-driven communications is the concept of Internet media types [2]. Media types are used to label representations so that it is known how the representation should be interpreted and how it is encoded. The centerpiece of a CoRE Application Description should be one or more media types.

Note: The terms media type and representation format are often used interchangeably. In this document, the term "media type" refers specifically to a string of characters such as "application/xml" that is used to label representations; the term "representation format" refers to the definition of the syntax and semantics of representations, such as XML 1.0 [12] or XML 1.1 [13].

A media type identifies a versioned series of representation formats (Section 2.2): a media type does not identify a particular version of a representation format; rather, the media type identifies the family, and includes provisions for version indicator(s) embedded in the representations themselves to determine more precisely the nature

of how the data is to be interpreted [20]. A new media type is only needed to designate a completely incompatible format [20].

Media types consist of a top-level type and a subtype, structured into trees [2]. Optionally, media types can have parameters. For example, the media type "text/plain; charset=utf-8" is a subtype for plain text under the "text" top-level type in the standards tree and has a parameter "charset" with the value "utf-8".

Media types can be further refined by

- o structured type name suffixes (e.g., "+xml" appended to the base subtype name; see Section 4.2.8 of RFC 6838 [2]),
- o a "profile" parameter (see Section 3.1 of RFC 6906 [24]),
- o subtype information embedded in the representations themselves (e.g., "xmlns" declarations in XML documents [11]),

or a similar annotation. An annotation directly in the media type is generally preferable, since subtype information embedded in representations can typically not be negotiated during content negotiation (e.g., using the CoAP Accept option).

In CoAP, media types are paired with a content coding [15] to indicate the "content format" [23] of a representation. Each content format is assigned a numeric identifier that can be used instead of the (more verbose) textual name of the media type in representation formats with size constraints. The flat number space loses the structural information that the textual names have, however.

The media type of a representation must be determined from in-band information (e.g., from the CoAP Content-Format option). Clients must not assume a structure from the application context or other out-of-band information.

IANA maintains a list of registered Internet media types at <http://www.iana.org/assignments/media-types>.

IANA maintains a list of registered structured suffixes at <http://www.iana.org/assignments/media-type-structured-suffix>.

IANA maintains a list of registered CoAP content formats at <http://www.iana.org/assignments/core-parameters>.

2.3. Links

As defined in RFC 8288 [6], a link is a typed connection between two resources. Additionally, a link is the primary means for a client to navigate from one resource to another.

A link is comprised of:

- o a link context,
- o a link relation type that identifies the semantics of the link (see Section 2.3.1),
- o a link target, identified by a URI, and
- o optionally, target attributes that further describe the link or the link target.

A link can be viewed as a statement of the form "{link context} has a {link relation type} resource at {link target}, which has {target attributes}" [6]. For example, the resource <http://example.com/> could have a "terms-of-service" resource at <http://example.com/tos>, which has a representation with the media type "text/html".

There are two special kinds of links:

- o An embedding link is a link with an additional hint: when the link is processed, it should be substituted with the representation of the referenced resource rather than cause the client to navigate away from the current resource. Thus, traversing an embedding link adds to the current state rather than replacing it.

The most well known example for an embedding link is the HTML element. When a Web browser processes this element, it automatically dereferences the "src" and renders the resulting image in place of the element.

- o A templated link is a link where the client constructs the link target URI from provided in-band instructions. The specific rules for such instructions are described by the representation format. URI Templates [3] provide a generic way to construct URIs through variable expansion.

Templated links allow a client to construct resource URIs without being coupled to the resource structure at the server, provided that the client learns the template from a representation sent by the server and does not have the template hard-coded.

2.3.1. Link Relation Types

A link relation type identifies the semantics of a link [6]. For example, a link with the relation type "copyright" indicates that the resource identified by the target URI is a statement of the copyright terms applying to the link context.

Relation types are not to be confused with media types; they do not identify the format of the representation that results when the link is dereferenced [6]. Rather, they only describe how the link context is related to another resource [6].

IANA maintains a list of registered link relation types at <http://www.iana.org/assignments/link-relations>.

Applications that don't wish to register a link relation type can use an extension link relation type [6]: a URI that uniquely identifies the link relation type. For example, an application can use the string "http://example.com/foo" as link relation type without having to register it. Using a URI to identify an extension link relation type, rather than a simple string, reduces the probability of different link relation types using the same identifiers.

2.3.2. Template Variable Names

A templated link enables clients to construct the target URI of a link, for example, when the link refers to a space of resources rather than a single resource. The most prominent mechanisms for this are URI Templates [3] and the HTML <form> element with a submission method of GET.

To enable an automated client to construct an URI reference from a URI Template, the name of the variable in the template can be used to identify the semantics of the variable. For example, when retrieving the representation of a collection of temperature readings, a variable named "threshold" could indicate the variable for setting a threshold of the readings to retrieve.

Template variable names are scoped to link relation types, i.e., two variables with the same name can have different semantics if they appear in links with different link relation types.

2.4. Forms

A form is the primary means for a client to submit information to a server, typically in order to change resource state.

A form is comprised of:

- o a form context,
- o a form relation type that identifies the semantics of the form (see Section 2.4.1),
- o a request method (e.g., PUT, POST, DELETE),
- o a submission URI,
- o a description of a representation that the server expects as part of the form submission, and
- o optionally, target attributes that further describe the form or the form target.

A form can be viewed as an instruction of the form "To perform a {form relation type} operation on {form context}, make a {request method} request to {submission URI}, which has {target attributes}". For example, to "update" the resource <http://example.com/config>, a client would make a PUT request to <http://example.com/config>. (In many cases, the target of a form is the same resource as the context, but this is not required.)

The description of the expected representation can be a set of form fields (see Section 2.4.2) or simply a list of acceptable media types.

Note: A form with a submission method of GET is, strictly speaking, a templated link, since it provides a way to construct a URI and does not submit a representation to the server.

2.4.1. Form Relation Types

A form relation type identifies the semantics of a form. For example, a form with the form relation type "create" indicates that a new item can be created within the form context by making a request to the resource identified by the target URI.

Similarly to extension link relation types, applications can use extension form relation types when they don't wish to register a form relation type.

2.4.2. Form Field Names

Forms can have a detailed description of the representation expected by the server as part of form submission. This description typically consists of a set of form fields where each form field is comprised

of a field name, a field type, and optionally a number of attributes such as a default value, a validation rule or a human-readable label.

To enable an automated client to fill out a form, the field name can be used to identify the semantics of the form field. For example, when controlling a smart light bulb, the field name "brightness" could indicate the field for setting the desired brightness of the light bulb.

Field names are scoped to form relation types, i.e., two form fields with the same name can have different semantics if they appear in forms with different form relation types.

The type of a form field is a data type such as "an integer between 1 and 100" or "an RGB color". The type is orthogonal to the field name, i.e., the type should not be determined from the field name even though the client can identify the semantics of the field from the name. This separation makes it easy to change the set of acceptable values in the future.

2.5. Well-Known Locations

Some applications may require the discovery of information about a host, known as "site-wide metadata" in RFC 5785 [4]. For example, RFC 6415 [19] defines a metadata document format for describing a host; similarly, RFC 6690 [22] defines a link format for the discovery of resources hosted by a server.

Applications that need to define a resource for this kind of metadata can register new "well-known locations". RFC 5785 [4] defines the path prefix `"/.well-known/"` in "http" and "https" URIs for this purpose. RFC 7252 [23] extends this convention to "coap" and "coaps" URIs.

IANA maintains a list of registered well-known URIs at <http://www.iana.org/assignments/well-known-uris>.

3. CoRE Application Descriptions

As applications are implemented and deployed, it becomes important to describe them in some structured way. This section provides a simple template for CoRE Application Descriptions. A uniform structure allows implementers to easily determine the components that make up the interface of an application.

The template below lists all components of applications that both the client and the server implementation of the application need to understand in order to interoperate. Crucially, items not listed in

the template are not part of the contract between clients and servers -- they are implementation details. This includes in particular the URIs of resources (see Section 4).

CoRE Application Descriptions are intended to be published in human-readable format by designers of applications and by operators of deployed application instances. Application designers may publish an application description as a general specification of all application instances, so that implementers can create interoperable clients and servers. Operators of application instances may publish an application description as part of the API documentation of the service, which should also include instructions how the service can be located and which communication protocols and security modes are used.

3.1. Template

The fields of the template are as follows:

Application name:

Name of the application. The name is not used to negotiate capabilities; it is purely informational. A name may include a version number or, for example, refer to a living standard that is updated continuously.

URI schemes:

URI schemes identifying the communication protocols that need to be understood by clients and servers. This information is mostly relevant for deployed instances of the application rather than for the general specification of the application.

Media types:

Internet media types that identify the representation formats that need to be understood by clients and servers. An application description must comprise at least one media type. Additional media types may be required or optional.

Link relation types:

Link relation types that identify the semantics of links. An application description may comprise IANA-registered link relation types and extension link relation types. Both may be required or optional.

Template variable names:

For each link relation type, variable names that identify the semantics of variables in templated links with that link relation type. Whether a template variable is required or optional is indicated in-band inside the templated link.

Form relation types:

Form relation types that identify the semantics of forms and, for each form relation type, the submission method(s) to be used. An application description may comprise IANA-registered form relation types and extension form relation types. Both may be required or optional.

Form field names:

For each form relation type, form field names that identify the semantics of form fields in forms with that form relation type. Whether a form field is required or optional is indicated in-band inside the form.

Well-known locations:

Well-known locations in the resource identifier space of servers that clients can use to discover information given the DNS name or IP address of a server.

Interoperability considerations:

Any issues regarding the interoperable use of the components of the application should be given here.

Security considerations:

Security considerations for the security of the application must be specified here.

Contact:

Person (including contact information) to contact for further information.

Author/Change controller:

Person (including contact information) authorized to change this application description.

Each field should include full citations for all specifications necessary to understand the application components.

4. URI Design Considerations

URIs [1] are a cornerstone of RESTful applications. They enable uniform identification of resources via URI schemes [7] and are used every time a client interacts with a particular resource or when a resource representation references another resource.

URIs often include structured application data in the path and query components, such as paths in a filesystem or keys in a database. It is common for many RESTful applications to use these structures not only as an implementation detail but also make them part of the

public REST API, prescribing a fixed format for this data. However, there are a number of problems with this practice [5], in particular if the application designer and the server owner are not the same entity.

In hypermedia-driven applications, URIs are therefore not included in the application interface. A CoRE Application Description must not mandate any particular form of URI substructure.

RFC 7320 [5] describes the problematic practice of fixed URI structures in detail and provides some acceptable alternatives.

Nevertheless, the design of the URI structure on a server is an essential part of implementing a RESTful application, even though it is not part of the application interface. The server implementer is responsible for binding the resources identified by the application designer to URIs.

A good RESTful URI is:

- o Short. Short URIs are easier to remember and cause less overhead in requests and representations.
- o Meaningful. A URI should describe the resource in a way that is meaningful and useful to humans.
- o Consistent. URIs should follow a consistent pattern to make it easy to reason about the application.
- o Bookmarkable. Cool URIs don't change [9]. However, in practice, application resource structures do change. That should cause URIs to change as well so they better reflect reality. Implementations should not depend on unchanging URIs.
- o Shareable. A URI should not be context sensitive, e.g., to the currently logged-in user. It should be possible to share a URI with third parties so they can access the same resource.
- o Extension-less. Some applications return different data for different extensions, e.g., for "contacts.xml" or "contacts.json". But different URIs imply different resources. RESTful URIs should identify a single resource. Different representations of the resource can be negotiated (e.g., using the CoAP Accept option).

5. Security Considerations

The security considerations of RFC 3986 [1], RFC 5785 [4], RFC 6570 [3], RFC 6838 [2], RFC 7320 [5], RFC 7595 [7], and RFC 8288 [6] are inherited.

All components of an application description are expected to contain clear security considerations. CoRE Application Descriptions should furthermore contain security considerations that need to be taken into account for the security of the overall application.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

- [1] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [2] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [3] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [4] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [5] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.
- [6] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

- [7] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<https://www.rfc-editor.org/info/rfc7595>>.

7.2. Informative References

- [8] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [9] Berners-Lee, T., "Cool URIs don't change", 1998, <<http://www.w3.org/Provider/Style/URI.html>>.
- [10] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [11] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <<http://www.w3.org/TR/2009/REC-xml-names-20091208>>.
- [12] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.
- [13] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., Yergeau, F., and J. Cowan, "Extensible Markup Language (XML) 1.1 (Second Edition)", World Wide Web Consortium Recommendation REC-xml11-20060816, August 2006, <<http://www.w3.org/TR/2006/REC-xml11-20060816>>.
- [14] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [15] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.

- [16] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [17] Fielding, R., "REST APIs must be hypertext-driven", October 2008, <<http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>>.
- [18] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/info/rfc7301>>.
- [19] Hammer-Lahav, E., Ed. and B. Cook, "Web Host Metadata", RFC 6415, DOI 10.17487/RFC6415, October 2011, <<https://www.rfc-editor.org/info/rfc6415>>.
- [20] Masinter, L., "MIME and the Web", draft-masinter-mime-web-info-02 (work in progress), January 2011.
- [21] Richardson, L. and M. Amundsen, "RESTful Web APIs", O'Reilly Media, ISBN 978-1-4493-5806-8, September 2013.
- [22] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [23] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [24] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <<https://www.rfc-editor.org/info/rfc6906>>.

Acknowledgements

Jan Algermissen, Mike Amundsen, Mike Kelly, Julian Reschke, and Erik Wilde provided valuable input on link and form relation types.

Thanks to Olaf Bergmann, Carsten Bormann, Stefanie Gerdes, Ari Keranen, Michael Koster, Matthias Kovatsch, Teemu Savolainen, and Bilhanan Silverajan for helpful comments and discussions that have shaped the document.

Some of the text in this document has been borrowed from [5], [6], [17], and [20]. All errors are my own.

This work was funded in part by Nokia.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 20, 2016

G. Selander
F. Palombini
Ericsson AB
K. Hartke
Universitaet Bremen TZI
L. Seitz
SICS Swedish ICT AB
March 19, 2016

Requirements for CoAP End-To-End Security
draft-hartke-core-e2e-security-reqs-00

Abstract

This document analyses threats to CoAP message exchanges traversing proxies and derives the security requirements for mitigating those threats.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 20, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Scope and Assumptions	3
3. Scenarios, Threats and Security Requirements	6
3.1. One Request - One Response	7
3.1.1. Processing Rules	8
3.1.2. Security Objectives	9
3.1.3. Threat Analysis and Mitigation	10
3.1.4. Security Requirements	13
3.2. One Request - Multiple Responses	14
3.2.1. Processing Rules	15
3.2.2. Security Objectives	16
3.2.3. Threat Analysis and Mitigation	16
3.2.4. Security Requirements	17
3.3. Multiple Requests - One Response	17
3.3.1. Processing Rules	19
3.3.2. Security Objectives	19
3.3.3. Threat Analysis and Mitigation	20
3.3.4. Security Requirements	24
3.4. Multiple Requests - Multiple Responses: Observe	25
3.4.1. Processing Rules	27
3.4.2. Security Objectives	27
3.4.3. Threat Analysis and Mitigation	27
3.4.4. Security Requirements	28
3.5. Multiple Requests - Multiple Responses: Publish-Subscribe	28
3.5.1. Processing Rules	30
3.5.2. Security Objectives	30
3.5.3. Threat Analysis and Mitigation	30
3.5.4. Security Requirements	33
4. Security Considerations	34
5. IANA Considerations	35
6. References	35
6.1. Normative References	35
6.2. Informative References	35
Acknowledgments	36
Authors' Addresses	36

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a Web application protocol designed for constrained nodes and networks [RFC7228].

CoAP uses Datagram Transport Layer Security (DTLS) [RFC6347] for security. At the same time, CoAP relies on proxies for scalability and efficiency. These proxies are specified to perform a number of operations on CoAP messages which requires DTLS to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the desired proxy functionality, but is also able to eavesdrop on or manipulate any part of the CoAP payload and metadata in transit between client and server or inject new CoAP messages without being protected or detected by DTLS.

One way to mitigate this threat is to secure CoAP communication at the application layer using an object-based security mechanism such as CBOR Encoded Message Syntax [I-D.ietf-cose-msg] instead of or in addition to the security mechanisms at the network layer or transport layer. Such a mechanism can provide "end-to-end security" at the application layer in contrast to the "hop-by-hop security" provided by DTLS.

This document analyses security requirements for CoAP requests and responses of sensor and actuator deployments involving proxies and other similar intermediaries. The analysis is based on identifying the assets associated to sensor- and actuator-based communication patterns and considering the potential threats executed through proxies to these assets. The threat analysis provides the basis for defining the security requirements that an end-to-end security mechanism for CoAP needs to meet.

1.1. Terminology

Readers are expected to be familiar with the terms and concepts described in [RFC7252].

2. Scope and Assumptions

This document presents a number of scenarios involving sensor and actuator communications over CoAP. Common to all scenarios is the presence of at least one CoAP intermediary, typically in the form of a proxy between a client requesting a resource and a server hosting a resource (see Figure 1). The proxy is responsible, for example, for reducing response time and network bandwidth use by serving responses from a cache or for enabling the client to make requests that it otherwise could not make.

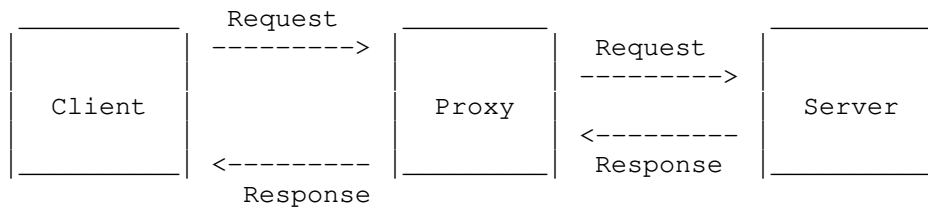


Figure 1: CoAP Message Exchanges Through A Proxy

The basic function of a proxy is to forward translated messages according to certain processing rules. For example:

- o Forward a message to the next proxy when the link is up
- o Only forward a request if there is no fresh cached response
- o Forward a new publication to all subscribing clients

In order to perform its function, a proxy may be required to read or change certain parts of a CoAP message as defined in [RFC7252]. For example, a forward proxy is defined to transform the Proxy-Uri option to Uri-Host, Uri-Port, Uri-Path and Uri-Query options. A proxy caching responses needs to read the Cache Key and is required to change the Max-Age option in the responses.

Since a proxy might not be fully trusted, a security solution is needed that protects the client, the server and the message exchanges against certain threats while still allowing the proxy to assume its normal functionality. The client and server are assumed to have a security association, but the proxy is neither assumed to have a security association with the client nor with the server.

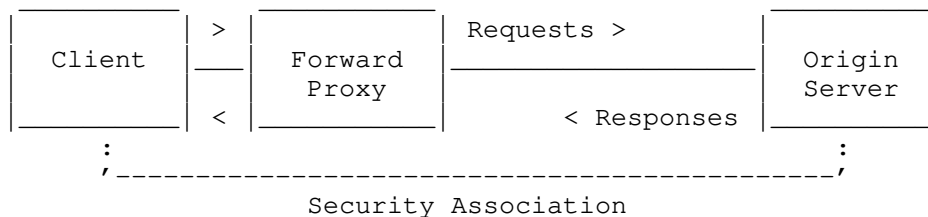


Figure 2: Security Association Between Client and Server

For a start, this document considers the following two cases: Forward proxies (as specified in [RFC7252]; Figure 2) and publish-subscribe brokers (as specified in [I-D.koster-core-coap-pubsub]; Figure 3).

The functionality assumed by these nodes is summarized in the respective scenarios analyzed in this document (Section 3).

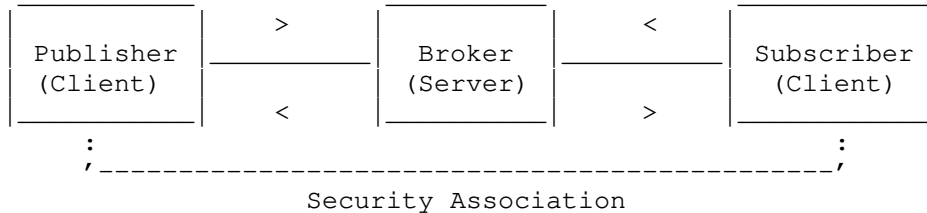


Figure 3: Security Association Between Publisher and Subscriber

[TODO: Reverse proxy and cross-protocol proxies will be added in a future version of this document.]

To identify the threats in scope, we first consider what assets need to be protected. In general, there are the following types of assets to protect:

- A1: The devices at the two ends, the data generated and stored in these devices, and their (often very constrained) system resources such as available memory, storage, processing capacity, and energy.
- A2: The physical environment of the devices fitted with sensors and actuators. Access to the physical environment is provided through CoAP resources that allow a remote entity to retrieve information about the physical environment (such as the current temperature) or to produce an effect on the physical environment (such as the activation of a heater).
- A3: The communication infrastructure linking the two devices (which often contains some very constrained parts) and the data stored in the message processing devices.

The scope of this document is to analyze threats executed through proxies and brokers, and this is only directly affecting the assets of type A3, e.g., if a proxy is dropping all messages.

However, the intermediary node may manipulate the messages exchanged between the endpoints and thereby have an impact also on the assets A1 and A2, for example: flooding a device with messages has impact on its system resources, and successful manipulation of an actuator command, carried in a message, has an impact on the physical environment. We therefore add a fourth asset, which is the main target being evaluated in this document:

A4: The messages exchanged between a client and a server, through the proxy. This includes the CoAP header and options in request and response messages (such as the requested method or the target URI) and the CoAP resource representations, encapsulated in the message payload.

A fully trusted proxy, handling unprotected messages, is an attractive target, since proxies are aggregation points for message flows (see Section 4) and they may be an easier target from the Internet than the sensors/actuators residing behind them. A proxy may become subject to intrusion or become infected by malware and perform the attacks of a man-in-the-middle. The attack vectors for compromising a proxy and the associated risks are out of scope for this document.

The scope of the threat analysis is restricted to threats from proxies to single client to server interactions. Threats resulting from collusion between multiple proxies are also out of scope (see Section 4).

On a high level, there are the following threats from proxies to consider:

- T1: The proxy illegitimately modifies a message.
- T2: The proxy illegitimately sends a message, including replay, flooding, etc.
- T3: The proxy illegitimately inhibits sending of a message, including delay, reordering, etc.
- T4: The proxy illegitimately reads part of a message.

To assess how such threats impact the assets, we need to specify the processing rules of the intermediary nodes in different scenarios and define the associated security objectives.

3. Scenarios, Threats and Security Requirements

In this section we consider a set of scenarios involving proxies and brokers, with different processing rules and security objectives. We study the associated threats and derive the security requirements for message transfer between client and server, in the different scenarios.

Note that, since CoAP was not designed for end-to-end security, solutions complying with these security requirements extend the applicability of CoAP beyond its original scope.

To simplify the analysis, the scenarios are structured according to how requests and responses are related to each other:

One Request - One Response

There is a one-to-one relation between request and response.

One Request - Multiple Responses

A request may have multiple responses, but each response is securely linked to a unique request.

Multiple Requests - One Response

One response may serve multiple requests, but each request has a single response.

Multiple Requests - Multiple Responses

One response may serve multiple requests, and each request may have multiple responses.

3.1. One Request - One Response

In this scenario we study use cases where it is important that a response sent from one endpoint is the response to a particular request to that endpoint. Many security critical use cases require that responses are in this way "securely linked" to requests, such as alarm status retrieval and actuator command confirmation.

In this scenario there must be a unique response for each request.

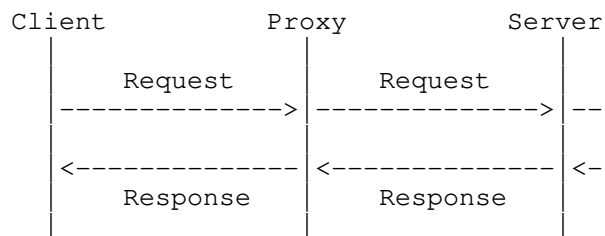


Figure 4: Message Flow with a Unique Response for Each Request

Example: Alarm status retrieval

Figure 4 can be seen as an illustration of a message exchange for a client requesting the alarm status (e.g., GET /alarm_status) from a server. Since the client wants to ensure that the alarm status received is reflecting the current alarm status and not a cached or spoofed response to the same resource, it must be able to verify that the received response is a response to this

particular request made by the client. Therefore the response must be securely linked to the request.

Example: Actuation confirmation

Another example for which Figure 4 serves as illustration is the confirmation of an actuator request. In this case a client, say in an industrial control system, requests a server that a valve should be turned to a certain level, e.g. PUT /valve_42/level with payload "3". In order for the client to correctly evaluate the result of a potential changed valve level, it is important that the client gets a confirmation how the server responded to the requested change, e.g., whether the request was performed or not. Again, the client wants to ensure that the response is reflecting the result of this particular actuation request made by the client and not a cached or spoofed response. Therefore the response must be securely linked to the request.

Functional Requirement:

- o Since each response is intended to be securely linked to a particular request, the response must not be used with any other request. Hence, as much as possible of the caching functionality must be inhibited. Therefore the CoAP option Max-Age of the responses is set to 0 (see Section 5.7.1 of [RFC7252]).

3.1.1. Processing Rules

In this scenario, the desired proxy functionality is to forward a translated request to the determined destination. There are two modes of operation for requests: Either using the Proxy-Uri option (PR1.1) or using the Proxy-Scheme option together with the Uri-Host, Uri-Port, Uri-Path and Uri-Query options (PR1.2).

PR1.1 The Proxy-Uri option contains the request URI including request scheme (e.g. "coaps://"); the Proxy-Scheme and Uri-* options are not present.

If the proxy is configured to forward requests to another proxy, then it keeps the Proxy-Uri option; otherwise, it splits the option into its components, adds the corresponding Uri-* options and removes the Proxy-Uri option. Then it makes the request using the request scheme indicated in the Proxy-Uri.

PR1.2 The Proxy-Scheme option and the Uri-* options together contain the request URI; the Proxy-Uri option is not present.

If the proxy is configured to forward requests to another forwarding proxy, then it keeps the Proxy-Scheme and Uri-* options; otherwise, it removes the Proxy-Scheme option. Then it makes the request using the request scheme indicated in the removed Proxy-Scheme option.

PR1.3 Responses are forwarded by the proxy, without any modification.

3.1.2. Security Objectives

In this scenario there is a unique response for each request, so the client should be able to verify that a certain response is made in response to a specific request sent by the client.

The server should be able to verify that the proxy only has performed the message modifications intended by the client according to the processing rules.

The proxy should be prevented from reading or making modifications to messages apart from what is necessary to perform the processing rules (cf. [RFC7258]).

The security objectives are:

- S01.1 The server is able to verify that a received request originates from a client with which it has a security association, and that the request has not been received before.
- S01.2 The server is able to verify that the received request either has not been altered in transfer, or that the request is modified according to the processing rule PR1.1 or PR1.2 (Section 3.1.1).
- S01.3 The server is able to protect the response such that only authorized clients can read the response.
- S01.4 The client is able to verify that the received response originates from the requested server and resource, that it has not been altered in transfer, and that it was generated as the unique response to the request.
- S01.5 The proxy is only able to read data needed to perform the processing rules.

3.1.3. Threat Analysis and Mitigation

We now list potential threats and discuss candidate mitigation mechanisms.

3.1.3.1. T1: The proxy illegitimately modifies a message

T1:1.1 The proxy forwards a request with modified payload

This threat can be mitigated with integrity protection of payload.

T1:1.2 The proxy forwards a response with modified payload

This threat can be mitigated with integrity protection of payload.

T1:1.3 The proxy forwards a request with modified CoAP option

Note that the proxy is entitled to change certain options by processing rules PR1.1 and PR1.2. Since the change is predictable, the effective request URI can be integrity protected by the client and verified by the server. The other CoAP options in the request can be integrity protected.

T1:1.4 The proxy forwards a response with modified CoAP option

This threat can be mitigated with integrity protection of CoAP options. Since Max-Age is set to 0 the proxy is not entitled to change any options in the response so they can all be integrity protected.

T1:1.5 The proxy forwards a request with changed CoAP header fields

The proxy is entitled to change certain header fields (e.g., the token) as part of its normal operations. Malicious changes to message layer parameters may cause a denial-of-service, equivalent of dropping a message or sending spoofed messages. This is difficult to mitigate. However, changing the CoAP header Code (e.g., from GET to DELETE) may result in an error or wrong interpretation of the request which can have other security implications. A change to the Version header field may result in security errors in the interaction between different versions of CoAP. These threats can be mitigated by integrity protecting the Code and Version header fields.

T1:1.6 The proxy forwards a response with changed CoAP header fields

Similar to previous threat. Some aspects of this threat can be mitigated by integrity protecting the Code and Version header fields.

T1:1.7 The proxy forwards a different request

If the forwarded request is from another client it can be mitigated by having different security associations with different clients. If the forwarded request is from the same client but with differences in payload, options or header, then this coincides with previously listed threats. A proxy sending old requests (or reordering requests) from the same client to the same server resource can be mitigated by integrity protecting a freshness parameter (timestamp, counter, etc.) from which the order of requests can be deduced (replay/reordering protection).

T1:1.8 The proxy forwards a different response

By integrity protecting uniquely identifying information of the request in the response, the client can verify that the response was generated in reply to a particular request.

3.1.3.2. T2:The proxy illegitimately sends a message

T2:1.1 The proxy sends a request to the server without a previous request from the client

This threat may be mitigated with integrity- and replay protection.

T2:1.2 The proxy sends a response to the client without a previous response from the server

Error messages from the proxy such as 5.02 (Bad Gateway) originate from the proxy. A proxy maliciously sending error messages is a denial-of-service attack similar to not forwarding a message (T3:1.1) and is difficult to mitigate. However, responses claiming to be from the server may be mitigated with integrity protection uniquely identifying information of the request.

T2:1.3 A proxy sends a number of messages for the purpose of flooding client or server

By verifying the integrity, the client and server may mitigate certain flooding attacks. The server can use the replay/reordering protection to verify which messages are

legitimate and the client can verify if a message is a response to a previously sent request.

3.1.3.3. T3:The proxy illegitimately inhibits sending of a message

T3:1.1 The proxy does not forward a message

This is a denial-of-service attack. While these kind of threats may be difficult to mitigate, applications should have a readiness for this kind of issues and a client is able to detect a missing response.

T3:1.2 The proxy delays forwarding of a received message

Delayed forwarding may be a denial-of-service attack, similar to not forwarding. Certain delays may be legitimate, so they may be difficult to detect and mitigate. However, delayed requests and responses can also be used in attacks against actuators; see [I-D.mattsson-core-coap-actuators]. These attacks can be performed by an on-path attacker and are not restricted to proxies. The proposed mitigation is based on verifying the timeliness of the request, for example, by using time stamps or with an additional round-trip. These mitigations can be supported by a new CoAP option containing time stamp or binding the response in a first round-trip to a request of the second, as specified in [I-D.mattsson-core-coap-actuators]. By integrity protecting that new CoAP option, the threat can be mitigated.

T3:1.3 The proxy reorders the requests

This threat may be mitigated with the server integrity protecting a freshness parameter from which the order of requests can be deduced.

T3:1.4 The proxy reorders the responses

This threat may be mitigated with the server integrity protecting information specifying to which request a response belongs.

3.1.3.4. T4:The proxy illegitimately reads part of a message

T4:1.1 The proxy reads a representation/payload

This threat can be mitigated with encryption of the payload.

- T4:1.2 The proxy infers information about the nature and state of the resource request/response from CoAP options

The proxy only needs to read the Uri-Host/Uri-Port and Proxy-Uri/Proxy-Scheme options of a request. The information revealed by these parameters is public on network layer. The proxy only needs to read Max-Age of the response, which is set to 0 as indicated in the functional requirements. This threat can be mitigated by encrypting all other options.

- T4:1.3 The proxy infers information about the nature and state of the resource request/response from CoAP header fields

The header fields needs to be transferred in plain text to allow normal CoAP operations. The Code parameter reveals information about what RESTful action is requested. This information leakage is difficult to mitigate.

- T4:1.4 The proxy reads and stores all message exchanges and can deduce information about the entire history of the corresponding interactions

This threat can be mitigated with encrypting as much as possible of the data transferred between client and server. The case of long term key compromise can be mitigated with forward secrecy.

3.1.4. Security Requirements

This section contains the security requirements and non-requirements for this scenario. For each requirement and non-requirement the associated threats are listed. The security requirements are:

- R1.1 The server must authenticate a message coming from a requesting client (T1:1.1, T1:1.3, T1:1.5, T2:1.1).
- R1.2 The server must verify that it has not received this request previously (T1:1.7, T3:1.3).
- R1.3 The client must verify that the received response originates from the requested server (T1:1.2, T1:1.4, T1:1.6, T2:1.2).
- R1.4 The client must verify that a response corresponds uniquely to a previous request that the client has made (T1:1.8, T3:1.4).
- R1.5 The payload must be integrity protected and encrypted between client and server (T1:1.1-6, T4:1.1, T2:1.3, T4:1.1, [RFC7258]).

- R1.6 The CoAP options except Uri-* and Proxy-* must be integrity protected in the request. The effective request URI must be integrity protected in the request (T1:1.3).
- R1.7 All CoAP options in the response must be integrity protected. Max-Age must be set to 0 (T1:1.4).
- R1.8 The CoAP options Uri-Host/Port and Proxy-Uri/Scheme of the request must not be encrypted. The Max-Age option of the response must not be encrypted. All other options must be encrypted (T4:1.2).
- R1.9 The CoAP header fields Version and Code must be integrity protected in requests and responses. All other header fields must not be integrity protected. The header fields must not be encrypted (T1:1.5, T4:1.3).
- R1.10 The communication protocol must provide forward secrecy (T4:1.4).

The security non-requirements of this scenario are:

- NR1.1 The proxy may drop messages without the endpoint being able to infer that the message is lost due to the proxy (T3:1.1).
- NR1.2 The proxy may delay messages without being detected (T3:1.1, T3:1.2).
- NR1.3 The proxy may read the CoAP header including message layer parameters and Code, revealing the kind of RESTful action being requested and the response code (T4:1.3).

3.2. One Request - Multiple Responses

In this scenario we study use cases where it is important that a response is securely linked to a request as in the previous scenario, but where there may be multiple responses for each request. This functionality protects communication-constrained servers from repeated requests from the same client and thus saves system resources and bandwidth. This is useful in security critical monitoring scenarios where time synchronization cannot be guaranteed.

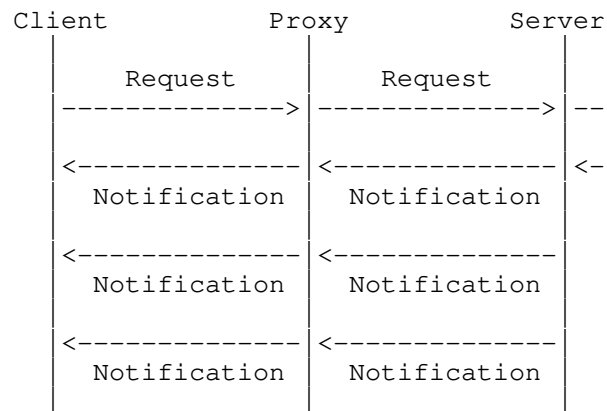


Figure 5: Message Flow of a Notification

Example: Secure parameter monitoring

Figure 5 can be seen as an illustration of a message exchange for a client monitoring an important parameter measured by the server, e.g., in a medical or process industry application. The client makes a subscription request for a resource and the server responds with notifications, thereby providing updates to the parameter in regular time intervals.

The client wants to ensure that first received notification reflects the current parameter value and that subsequent notifications are timely updates of the initial request. Since notifications may be lost or reordered, the client needs to be able to verify the order of the messages, as sent by the server. By monitoring the received messages and the time they are received, the client can detect missing notifications and take appropriate action.

Functional Requirement:

- o The same functional requirement apply as in the previous scenario (Section 3.1).

3.2.1. Processing Rules

The processing rules are identical to PR 1.1 - 1.3 of the previous scenario (Section 3.1.1).

3.2.2. Security Objectives

The security objectives are similar to the previous scenario. Each response maps to a unique request, but there may be multiple responses to one request. By ordering the responses, each message in this exchange can be made unique.

The security objectives of the previous scenario (Section 3.1.2) are valid except for S01.4 which is replaced by the following objectives:

S02.1 The client is able to verify that the received response originates from the requested server and resource, that it has not been altered in transfer, and that it was generated as one in a sequence of responses to the request.

S02.2 The client is able to verify the order of the responses and if a response is missing.

3.2.3. Threat Analysis and Mitigation

The threat analysis from the previous scenario carries over with a few exceptions.

3.2.3.1. T1:The proxy illegitimately modifies a message

Similar conclusions apply as in the previous scenario (Section 3.1.3.1). However, note that in T1:1.8, a proxy may maliciously reorder the responses to the same request without being detected. The mitigation specified in the previous scenario (that the client verifies the response is linked to the request) is not sufficient since there may be multiple responses.

However, analogous to how requests are protected against replay/reordering in the previous scenario, by additionally integrity protecting a parameter from which the order of responses can be deduced, this threat can be mitigated.

3.2.3.2. T2:The proxy illegitimately sends a message

Similar conclusions apply as in the previous scenario (Section 3.1.3.2). T2:1.3 can be mitigated with the additional replay/reordering protection of responses as mentioned in Section 3.2.3.1.

3.2.3.3. T3:The proxy illegitimately inhibits sending of a message

Similar conclusions apply as in the previous scenario (Section 3.1.3.3). T3:1.4 can be mitigated with the additional replay/reordering protection of responses as mentioned in Section 3.2.3.1.

3.2.3.4. T4:The proxy illegitimately reads part of a message

The same conclusions apply as in the previous scenario (Section 3.1.3.4).

3.2.4. Security Requirements

The security requirements of the previous scenario (Section 3.1.4) are valid except for R1.4 which is replaced by the following requirements:

- R2.1 The client must verify that a response corresponds to a previous request that the client has made (T1:1.8, T3:1.4).
- R2.2 The client must verify that it has not received this response previously and whether responses for the same request are received in the wrong order (T1:1.8, T3:1.3).

3.3. Multiple Requests - One Response

In this scenario we study caching: how a proxy may serve the same cached response to multiple clients requesting the same resource.

The caching functionality protects communication-constrained servers from repeated requests for the same resources, possibly originating from different clients. This saves system resources, bandwidth, and round-trip time.

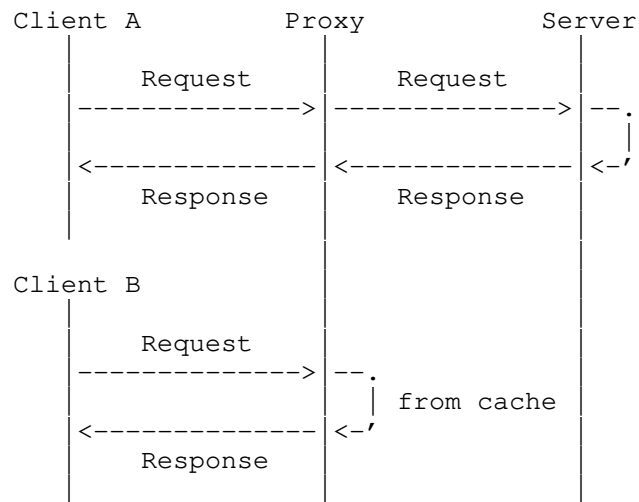


Figure 6: Message Flow for Cached Responses

In Figure 6, Client A requests the proxy to make a certain request to the server and to return the server's response. The proxy services the request by making a request message to the server according to the processing rules. If the server returns a cacheable response, then the proxy stores the response in its cache, performs any necessary translations, and forwards it to the client. Later, client B makes an equivalent request to the proxy that the proxy services by returning the response from its cache.

Cacheable responses are 2.05 (Content) responses and all error responses.

Functional Requirements:

- o The proxy must be able to store cacheable responses in a cache. This requires the proxy to read the CoAP header, options, and payload and to compute the cache key for a request.
- o The proxy must be able to return a fresh response from its cache without contacting the server.
- o The proxy must be able to perform validation on a request by a client and a request validation to the server (see Section 5.6.2 of [RFC7252]).

3.3.1. Processing Rules

The proxy complies with the forwarding rules PR1.1 - 1.3 (Section 3.1.1) and the rules below. The rules below have priority.

- PR3.1 If the proxy receives a request where the cache key matches that of a cached fresh response, then the proxy discards the request and replies with that response, else it makes a translated request.
- PR3.2 The proxy caches and forwards cacheable responses. If there is already a response in the cache with the cache key of the corresponding request, then the old response in the cache is marked as stale.
- PR3.3 If the proxy receives a request that contains an ETag option and the proxy has a fresh response with the same cache key and ETag, then the proxy replies to the request with a 2.03 (Valid) response without payload, else it forwards a translated request.
- PR3.4 The proxy updates the Max-Age option according to the Max-Age associated with the resource representation it receives, decreasing its value to reflect the time spent in the cache.
- PR3.5 If the request contains an Accept option and if there is a fresh response that matches the cache key for the corresponding request except for the Accept option, and if the Content-Format of the response matches that of the Accept option, then the proxy forwards the cached response to the requesting client.

3.3.2. Security Objectives

A caching proxy has an active role in the resource request/response procedure, so it is not surprising that it is necessary to make a trade-off between caching functionality and the protection of client-server interaction. Comparing with the scenario in Section 3.1, most of the security objectives in Section 3.1.2 cannot be met:

- o The caching functionality decouples responses from requests. This implies that a client is not able to verify that a received response is generated by the server in response to a specific request.
- o A client may receive a response without the server being aware that the client has made a request. A proxy could proactively generate requests or observe resources in order to keep the cache

up-to-date. Thus the server cannot in general verify that a request originates from a client as a precondition to provide a response.

Since a proxy can autonomously make requests for resource representations and there is no security association between proxy and server, the server cannot verify those requests. If a request needs to be verified then the solution to the scenario in Section 3.1 can be re-used. Therefore we do not consider the protection of requests and focus here on enabling the caching functionality and providing security to cacheable resource representations.

The security objectives for this scenario are:

- S03.1 The client is able to verify that a received response contains a resource representation to a requested server and resource, and that it has not been altered between server and client.
- S03.2 The client is able to verify that a received resource representation is fresh.
- S03.3 The server is able to protect a resource representation such that only authorized clients can read the representation.

3.3.3. Threat Analysis and Mitigation

We now list potential threats and discuss candidate mitigation mechanisms.

3.3.3.1. T1:The proxy illegitimately modifies a message

- T1:3.1 The proxy forwards a request with modified payload

Out of scope of the security objectives.

- T1:3.2 The proxy forwards a response with modified payload

This threat that may be mitigated with integrity protection of resource representation.

- T1:3.3 The proxy forwards a request with modified CoAP options

Out of scope of the security objectives.

- T1:3.4 The proxy forwards a response with modified CoAP options

This is not necessarily a threat. For example, a proxy is entitled to change Max-Age. However, changing Content-

Format may result in an error or the wrong interpretation of a representation. That kind of threat may be mitigated by securely associating resource information (such as Content-Format) to the representation in the response.

Tl:3.5 The proxy forwards a request with changed CoAP header fields

As mentioned in Section 3.1, this is not necessarily a threat and it is not in scope of the security objectives to mitigate.

Tl:3.6 The proxy forwards a response with changed CoAP header fields

This is not necessarily a threat, since message layer parameters may be changed by a proxy. A change of Code in the response may be misinterpreted. But as long as the responses allow verification of resource information, such a change will be detected. Thus this threat is mainly a denial-of-service. Threats arising from modification of Version are difficult to predict. A future version of CoAP must consider security implications of a proxy manipulating the version number.

Tl:3.7 The proxy forwards a request different from the translated request

Out of scope of the security objectives.

Tl:3.8 The proxy forwards a response to a non-equivalent request

If the response is from another server, then it can be mitigated by having different security associations with different servers. If the response is that of another resource of the same server, it can be mitigated by having different security associations of different resources, or by securely associating a resource identifier to the representation in the response. If the response is from the right server and resource, then the modifications of payload, options and header are considered previously.

Tl:3.9 The proxy forwards an old response to the same resource

This is not necessarily a threat. The proxy is supposed to send a cached response, if fresh. However, if the proxy serves a stale response and manipulates the Max-Age option, then it may trick the client into believing that this is a fresh response. Since the proxy is entitled to make such

changes, this is not possible to prevent. The server may however provide other freshness information (timestamp, counter, etc.) integrity protected together with the resource representation and associated resource information from which the client may infer that Max-Age is not correct. Note that in case time synchronization cannot be assumed the information about age is limited to the order of the responses.

- T1:3.10 The proxy maliciously serves a 2.03 (Valid) response to a request with an ETag option

This is not possible to prevent, since the proxy is entitled to perform such operation without involving the server. [TODO: Since the response must not include a payload (see Section 5.9.1.3 of RFC 7252), it is not clear how a server could enforce the proxy to include any integrity protected freshness information unless we define new proxy processing rules.]

- T1:3.11 The proxy colludes with a legitimate client having access to the key used to generate and verify Message Authentication Codes (MAC) of responses/resource representations to generate a valid MAC.

This threat applies to responses containing a message authentication code (MAC) for integrity protecting the resource representation. The threat may be mitigated by the server digitally signing the representation with its private key instead of using a MAC.

3.3.3.2. T2:The proxy illegitimately sends a message

- T2:3.1 The proxy sends a request to the server without a previous request from the client

This is not necessarily a threat, since the proxy may want to keep the cache updated with fresh representations to allow short round-trip time. A proxy maliciously making requests for the purpose of gaining information about the resources may to some extent be mitigated by encryption, but encrypting data in the cache key has an impact on how the cache can perform its legitimate operation. This is out of scope for the security objectives.

- T2:3.2 The proxy sends a response to the client without a previous response from the server

This is not necessarily a threat, since the proxy is allowed to respond with a fresh, cached response. Other cases of responding inappropriately to a client request are covered in the previous section. The client can detect the case of receiving a response without having sent a request.

- T2:3.3 A proxy sends a number of messages for the purpose of flooding client or server

Considering that a proxy is entitled to make resource requests, it may be difficult to protect the server against this kind of denial-of-service attacks. As for responses, by verifying the integrity and freshness of requested information, the client may mitigate certain flooding attacks.

- 3.3.3.3. T3:The proxy illegitimately inhibits sending of a message

- T3:3.1 The proxy does not forward a message

This is not necessarily a threat. According to the processing rule, the proxy must not forward a request if there is a fresh cached response. If the proxy does not forward a request although there is no valid cache response or if the proxy does not propagate a response, then this is a denial-of-service attack. While these threats may be difficult to mitigate, missing messages are common in lossy environments so applications should be prepared for this kind of issue.

- T3:3.2 The proxy delays forwarding of a received message

Delayed forwarding may be a denial-of-service attack, similar to not forwarding. Certain delays may be legitimate, so it is difficult to detect and mitigate this. Delayed requests and responses can also be used in attacks against actuators as is discussed in Section 3.1, but that is out of scope for this scenario.

- T3:3.3 The proxy reorders the requests

Out of scope of the security objectives.

- T3:3.4 The proxy reorders the responses

This threat may be mitigated with the server integrity protecting a freshness parameter together with the response.

3.3.3.4. T4:The proxy illegitimately reads part of a message

T4:3.1 The proxy reads a representation/payload

This threat can be mitigated with encryption of the representation, and other potential payload data.

T4:3.2 The proxy infers information about the nature and state of the resource request/response from CoAP options and header fields.

The proxy needs to read the cache key for performing caching operations. Information leaking that can be inferred from such data cannot be prevented.

T4:3.3 The proxy reads and stores all message exchanges and can deduce information about the entire history of resource access.

Since the cache key and other metadata is not in scope of the security objectives, the mitigation is restricted to encrypting the resource representations. The case of long term key compromise would nevertheless reveal the history of the resource, but this can be mitigated with forward secrecy.

3.3.4. Security Requirements

This section contains the security requirements and non-requirements for the caching scenario. For each requirement and non-requirement the associated threats are listed. The security requirements are:

R3.1 The client must be able to verify that a received resource representation originates from the requested server (T1:3.2, T1:3.8).

R3.2 The client must be able to verify that a received representation is a representation of the resource requested by the client (T1:3.2, T1:3.4, T1:3.8).

R3.3 The client must be able to verify the content format of the representation (T1:3.4).

R3.4 The client must be able to detect that a received representation is fresh (T1:3.9, T3:3.4).

R3.5 The representation must be integrity protected and encrypted from the server to the client (T1:3.2, T1:3.11, T2:3.3, T4:3.1).

R3.6 To protect against the proxy colluding with an authorized client, asymmetric cryptography must be used (T1:3.11).

R3.7 The communication protocol must provide forward secrecy (T4:3.3).

The security non-requirements of the caching scenario are:

NR3.1 The request is not protected (see Security Objectives).

NR3.2 The header and options of the response are not protected (see Security Objectives, compare R3.3).

NR3.3 The proxy may eavesdrop on metadata (including the cache key) or by making requests on behalf of alleged clients (T2:3.1, T4:3.2).

NR3.4 The proxy may drop messages without the endpoint being able to infer that the message is lost due to the proxy (T3:3.1).

NR3.5 The proxy may delay messages without being detected (T3:3.2).

NR3.6 The client may not be able to verify validity information provided by proxy when using ETag (T1:3.10).

3.4. Multiple Requests - Multiple Responses: Observe

This scenario is about replicating a resource state from a server to a client. The client observes a resource and receives notifications which may be cached. The difference compared to the previous scenario (Section 3.3) is the capability to send multiple responses in reply to a single request. The difference compared to Section 3.2 is that in this scenario multiple clients may be served with the same response.

This functionality protects communication-constrained servers from repeated requests, which may come from different clients, when the resource is unchanged. This saves system resources and bandwidth.

In addition to multiple clients' requests being served by one response, each request may result in multiple responses.

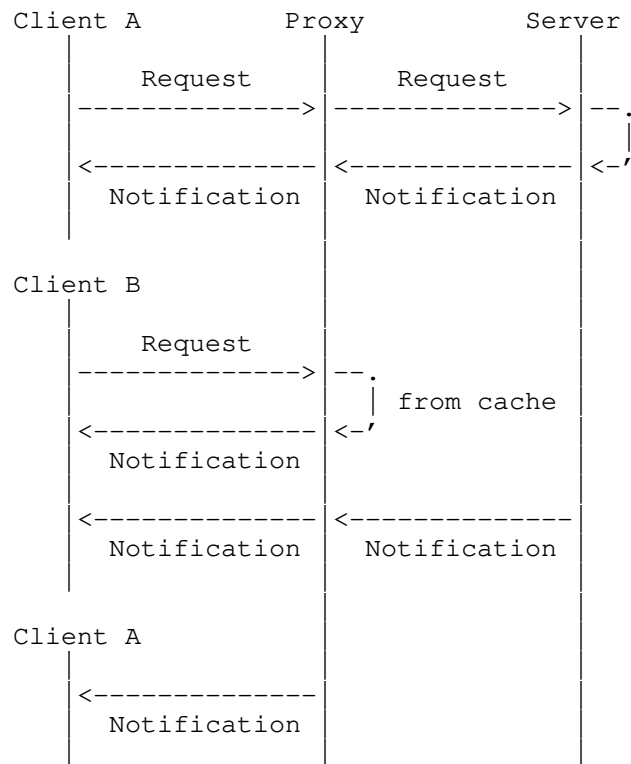


Figure 7: Message Flow for Observe with Multiple Observers

The server exposes an observable resource (e.g., the current reading of a temperature sensor). Multiple clients are interested in the current state of the resource and observe it using the CoAP resource observation mechanism [RFC7641]. The goal is to keep the state observed by the clients closely in sync with the actual state of the resource at the server. Another goal is to minimize the burden on the server by moving the task to fan out notifications to multiple clients from the server to the proxy.

Functional Requirements:

The functional requirements of the previous scenario (Section 3.3) apply, and additionally:

- o The proxy must be able to observe a resource on behalf of one or more clients.

- o When a client registers interest in a resource with the proxy, the proxy must be able to return a response containing the current state of the resource without contacting the server.

3.4.1. Processing Rules

The proxy complies with the processing rules PR3.1 - 3.5 of the previous scenario (Section 3.3.1). In addition, the following processing rules apply:

- PR4.1 If the proxy receives a notification from the server that is out of sequence (as indicated by the Observe option), then the proxy discards the notification. Otherwise, the proxy proceeds to notify the registered observers.
- PR4.2 When notifying an observer, the proxy modifies the Observe option to indicate the sequence of notifications from the proxy to the observer.

3.4.2. Security Objectives

The security objectives are identical to the previous scenario.

3.4.3. Threat Analysis and Mitigation

The threat analysis from the previous scenario carries over to this scenario.

3.4.3.1. T1:The proxy illegitimately modifies a message

The same conclusions apply as in the previous scenario (Section 3.3.3.1). For example in T1:3.4, a proxy may maliciously modify the Observe option to indicate a different order of notifications without being detected. However, the mitigation specified in the previous scenario applies: the server integrity protects a freshness parameter with the response.

3.4.3.2. T2:The proxy illegitimately sends a message

The same conclusions apply as in the previous scenario (Section 3.3.3.2).

3.4.3.3. T3:The proxy illegitimately inhibits sending of a message

The same conclusions apply as in the previous scenario (Section 3.3.3.3). The threat in T3:3.4 may be combined with manipulation of the Observe option, but the same mitigation as mentioned in (Section 3.4.3.1) applies.

3.4.3.4. T4: The proxy illegitimately reads part of a message

The same conclusions apply as in the previous scenario (Section 3.3.3.4).

3.4.4. Security Requirements

Since the security objectives and threat mitigations carry over from the previous scenario (Section 3.3), the same security requirements are valid (Section 3.3.4).

3.5. Multiple Requests - Multiple Responses: Publish-Subscribe

The intermediary node in the publish-subscribe scenario is a broker for messages from a publisher to subscriber. A subscriber subscribes to a "topic" and receives a publication. The broker fans out subsequent publications on that topic to all subscribers.

In this scenario a single request may result in multiple responses and a single response may reach multiple clients.

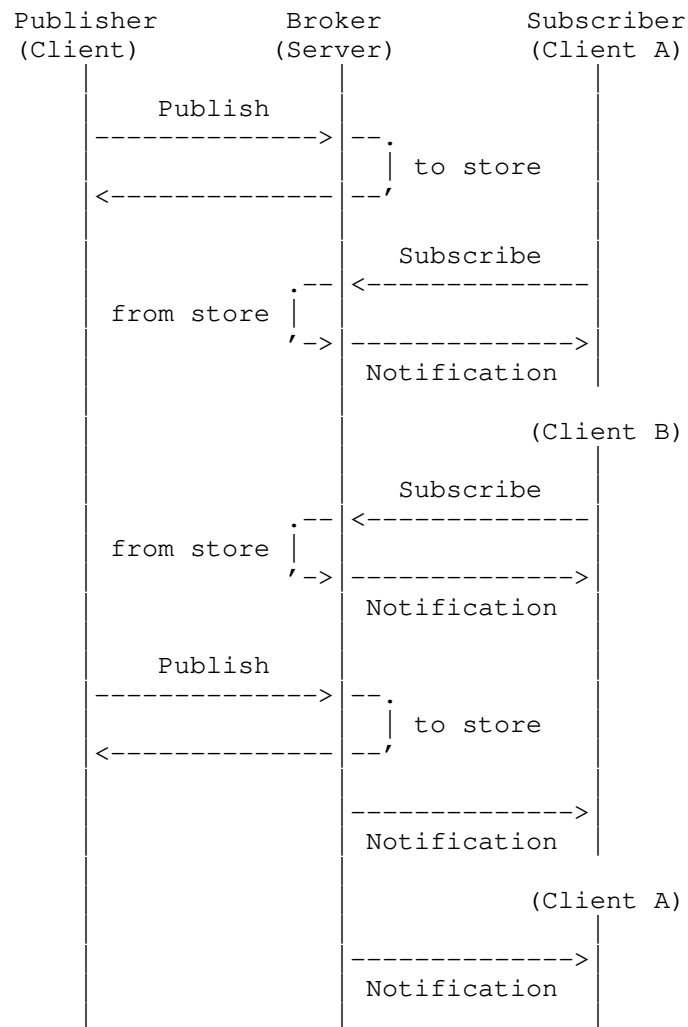


Figure 8: Message Flow for Publish-Subscribe

The broker maintains a number of topics that a publisher can publish to and a subscriber subscribe to. Topics are represented as URIs at the broker. Figure 8 illustrates the publication to a topic, implemented as a PUT request of a representation to a resource at the broker.

Subscribers can make a GET request with the Observe option to the topic URI at the broker in order to initiate the subscription on the topic. The broker provides a notification in the form of a stored representation as response to the request. Further publications of

representations to this URI are provided as notification responses to the subscription request.

Functional Requirement:

- o The publication must be able to be transferred in a PUT request from the publisher and in a GET response to the subscriber.

3.5.1. Processing Rules

- PR5.1 If the broker receives a subscription request to one of its resources, then the broker associates the requesting subscriber to the topic and responds with the current representation.
- PR5.2 If the broker receives a publication request to one of its resources, then the broker stores the received representation on the topic and responds with the representation to the associated subscribers of that topic.

Since we are focusing on end-to-end security between publisher and subscriber, the creation and deletion of topics and other endpoint-to-broker operations are out of scope.

3.5.2. Security Objectives

The security objectives for this scenario are:

- S05.1 A subscriber is able to verify that a received response contains a resource representation of a requested topic, that the publisher is authorized to publish to that topic, and that it has not been altered between publisher and subscriber.
- S05.2 A subscriber is able to verify that a received resource representation is fresh.
- S05.3 The publisher is able to protect a resource representation such that only authorized subscribers can read the representation.

3.5.3. Threat Analysis and Mitigation

We now analyze the potential threats relevant to this scenario.

3.5.3.1. T1:The broker illegitimately modifies a message

- T1:5.1 The broker responds to a subscriber request with a publication containing a modified payload

This threat may be mitigated with integrity protection of payload.

- T1:5.2 The broker responds to a subscriber request with a publication containing a modified CoAP option

Since the security objective is to protect the resource representation, only options in the GET response that influence the interpretation of the resource representations have an impact. A broker is entitled to change Max-Age and may do so maliciously. The broker is not entitled to change Content-Format, but may anyway do so maliciously. To mitigate these, the subscriber needs to be able to verify information about freshness and content format provided by the publisher.

- T1:5.3 The broker responds to a subscriber request with modified CoAP header fields

Since the security objective is to protect the resource representation, only header fields in the GET response that influence the interpretation of the resource representations have an impact. Changing of Code such as e.g. 2.05 (Content) to some error code is a denial-of-service.

- T1:5.4 The broker modifies the publication before or during storage

This threat is analogous to the previous threats and is mitigated in the same way.

- T1:5.5 The broker responds to a subscriber request with the wrong message

Modifications of payload, options, and header are considered previously. To mitigate wrong interpretation of a response resulting from a broker sending old messages or reordering messages from the same publisher to the same subscriber, the message may integrity protect a freshness parameter (timestamp, counter, etc.) from which the age/order can be deduced (replay/reordering protection).

- T1:5.6 The broker colludes with a legitimate subscriber having access to the key used to create Message Authentication Codes

(MAC) of publications in order to generate a valid MAC of a modified publication

This threat applies to publications containing a message authentication code (MAC) for integrity protecting the resource representation. The threat may be mitigated by the publisher digitally signing the representation with a private key instead of using a MAC.

3.5.3.2. T2:The broker illegitimately sends a message

T2:5.1 The broker sends a response to a subscriber request without a previous publication from the publisher

Most cases of responding inappropriately to a subscriber request are covered in the previous section. In general, authentication of publisher in combination with replay/reordering protection will mitigate this threat.

T2:5.2 A broker sends a number of messages for the purpose of flooding the subscriber

By verifying the integrity and freshness information, the subscriber may mitigate certain flooding attacks.

3.5.3.3. T3:The broker illegitimately inhibits sending of a message

T3:5.1 The broker does not store or forward a publication

This is a denial-of-service attack. While these threats may be difficult to mitigate, missing messages are common in lossy environments so applications should have a readiness for this kind of issue.

T3:5.2 The broker does not respond to a publication request

This may be a denial-of-service attack on the publisher. While such a threat may be difficult to mitigate, missing messages are common in lossy environments so applications should have a readiness for this kind of issue.

T3:5.3 The broker delays forwarding of a received publication

Delayed forwarding may be a denial-of-service attack, similar to not forwarding. Certain delays may be legitimate, so it may be difficult to detect and mitigate.

T3:5.4 The broker reorders the publications

This threat may be mitigated by the publisher integrity protecting the message and including a freshness parameter.

3.5.3.4. T4:The broker illegitimately reads part of a message

T4:5.1 The broker reads a representation/payload

This threat can be mitigated with encryption of the representation and other potential payload data

T4:5.2 The broker infers information about the nature and state of the publication from CoAP options and header fields.

This metadata is not in scope of confidentiality. Information leaking that can be inferred from such data cannot be prevented.

T4:5.3 The broker reads and stores all publications and can deduce information about the entire history of the publications and subscriptions

Since the protection of metadata related to subscription and publication is not in scope of the security objectives, the mitigation is restricted to encrypting the resource representations. The case of long term key compromise would nevertheless reveal the history of a publication, but this can be mitigated with forward secrecy.

3.5.4. Security Requirements

This section contains the security requirements and non-requirements for the publish-subscribe scenario. For each requirement and non-requirement the associated threats are listed. The security requirements are:

R5.1 The subscriber must be able to verify that a received resource representation originates from an authorized publisher (T1:5.1, T2:5.1).

R5.2 The subscriber must be able to verify that a received representation is a representation of the resource requested by the subscriber (T1:5.1, T1:5.4, T1:5.5, T1:5.6).

R5.3 The subscriber must be able to verify the content format of the representation (T1:5.2)

- R5.4 The subscriber must be able to detect that the received resource representation is older than a previously received representation of this resource (T1:5.5, T2:5.1, T3:5.4).
- R5.5 The representation must be integrity protected and encrypted from publisher to subscriber (T1:5.1, T1:5.5, T2:5.2, T4:5.1).
- R5.6 To protect against the proxy colluding with an authorized subscriber, asymmetric cryptography must be used (T1:5.6).
- R5.7 The communication protocol must provide forward secrecy (T4:5.3).

The security non-requirements of the pub-sub scenario are:

- NR5.1 The subscription request is not protected (see Security Objectives).
- NR5.2 The header and options of the notification response are not protected (see Security Objectives, compare R5.3).
- NR5.3 The broker may change and eavesdrop on certain metadata without being detected (T1:5.2, T1:5.3, T4:5.2).
- NR5.4 The broker may drop messages without being detected (T3:5.1, T3:5.2).
- NR5.5 The broker may delay messages without being detected (T3:5.3).

4. Security Considerations

A proxy or intermediary may be an aggregation point for message flows. Therefore it is an attractive target, both from a security and privacy point of view.

Unless the security mechanisms provide forward secrecy, a compromise of long term keying material means that an attacker can decrypt all previously sent information and can be directly used for any kind of manipulation of the cyber-physical system.

Therefore the key exchange mechanism used for establish keys to use with application layer security must provide forward secrecy.

Intermediary nodes are aggregation points also for metadata and therefore valuable targets for signal intelligence agencies. Pervasive monitoring is an attack [RFC7258] and the effect of collecting and correlating information from multitude of proxies must be mitigated.

Related to this, it is needed to delete all historical information from all nodes handling the plaintext data and metadata, in order to avoid information leakage. The impact of this on the intermediary nodes can be limited by confidentiality protecting as much as possible between the endpoints.

5. IANA Considerations

This document includes no request to IANA.

6. References

6.1. Normative References

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<http://www.rfc-editor.org/info/rfc7258>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

6.2. Informative References

- [I-D.ietf-cose-msg] Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-10 (work in progress), February 2016.
- [I-D.koster-core-coap-pubsub] Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-04 (work in progress), November 2015.
- [I-D.mattsson-core-coap-actuators] Mattsson, J., Fornehed, J., Selander, G., and F. Palombini, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-00 (work in progress), October 2015.

- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.

Acknowledgments

The authors wish to thank John Mattsson and Ari Keranen for reviewing early versions of the draft.

Authors' Addresses

Goeran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen 28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Ludwig Seitz
SICS Swedish ICT AB
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig@sics.se

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 30, 2018

G. Selander
F. Palombini
Ericsson AB
K. Hartke
Universitaet Bremen TZI
July 29, 2017

Requirements for CoAP End-To-End Security
draft-hartke-core-e2e-security-reqs-03

Abstract

This document analyses threats to CoAP message exchanges traversing proxies and derives security requirements for mitigating those threats.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 30, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Assets and Scope	4
1.2. Terminology	5
2. Proxying	6
2.1. Threats and Security Requirements	7
2.1.1. Client-side	7
2.1.1.1. Threat 1: Spoofing	8
2.1.1.2. Threat 2: Delaying	9
2.1.1.3. Threat 3: Withholding	9
2.1.1.4. Threat 4: Flooding	9
2.1.1.5. Threat 5: Eavesdropping	9
2.1.1.6. Threat 6: Traffic Analysis	9
2.1.2. Server-side	11
2.1.2.1. Threat 1: Spoofing	12
2.1.2.2. Threat 2: Delaying	12
2.1.2.3. Threat 3: Withholding	12
2.1.2.4. Threat 4: Flooding	12
2.1.2.5. Threat 5: Eavesdropping	13
2.1.2.6. Threat 6: Traffic Analysis	13
2.2. Solutions	14
2.2.1. Forwarding	15
2.2.1.1. Examples	15
2.2.1.2. Functional Requirements	17
2.2.1.3. Processing Rules	17
2.2.1.4. Authenticity	17
2.2.1.5. Confidentiality	19
2.2.2. Caching	19
2.2.2.1. Examples	19
2.2.2.2. Functional Requirements	21
2.2.2.3. Processing Rules	21
2.2.2.4. Authenticity	22
2.2.2.5. Confidentiality	23
3. Publish-Subscribe	24
3.1. Threats and Security Requirements	24
3.1.1. Subscriber-side	24
3.1.1.1. Threat 1: Spoofing	26
3.1.1.2. Threat 2: Delaying	27
3.1.1.3. Threat 3: Withholding	27
3.1.1.4. Threat 4: Flooding	27
3.1.1.5. Threat 5: Eavesdropping	27
3.1.1.6. Threat 6: Traffic Analysis	27
3.1.2. Publisher-side	27
3.2. Solutions	28
3.2.1. Brokering	28
3.2.1.1. Functional Requirements	30
3.2.1.2. Processing Rules	30

3.2.1.3. Authenticity	30
3.2.1.4. Confidentiality	30
4. Security Considerations	30
5. IANA Considerations	30
6. References	31
6.1. Normative References	31
6.2. Informative References	31
Acknowledgments	32
Authors' Addresses	32

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a Web application protocol designed for constrained nodes and networks [RFC7228]. CoAP makes use of Datagram Transport Layer Security (DTLS) [RFC6347] for security. At the same time, CoAP relies on proxies for scalability and efficiency. Proxies reduce response time and network bandwidth use by serving responses from a shared cache or enable clients to make requests that these otherwise could not make.

CoAP proxies need to perform a number of operations on requests and responses to fulfill their purpose, which requires the DTLS security associations to be terminated at each proxy. The proxies therefore do not only have access to the data required for performing the desired functionality, but are also able to eavesdrop on or manipulate any part of the CoAP payload and metadata exchanged between client and server, or inject new CoAP messages without being protected or detected by DTLS.

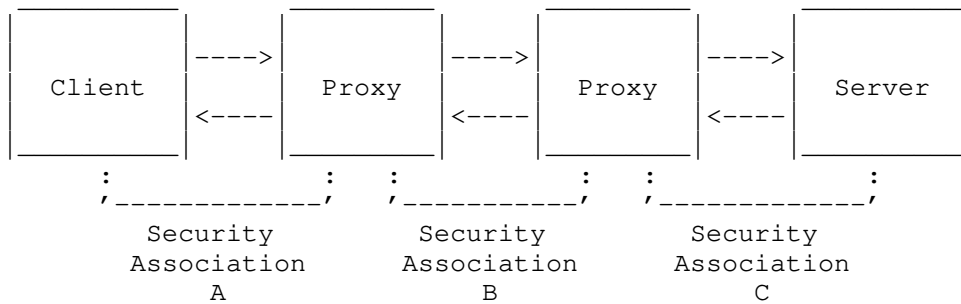


Figure 1: Hop-by-Hop Security

One way to mitigate this threat is to secure CoAP communication at the application layer using an object-based security mechanism such as CBOR Object Signing and Encryption (COSE) [RFC8152] instead of or in addition to the security mechanisms at the network layer or transport layer. Such a mechanism can provide "end-to-end security"

at the CoAP layer (Figure 2) in contrast to the "hop-by-hop security" that DTLS provides at the CoAP layer (Figure 1).

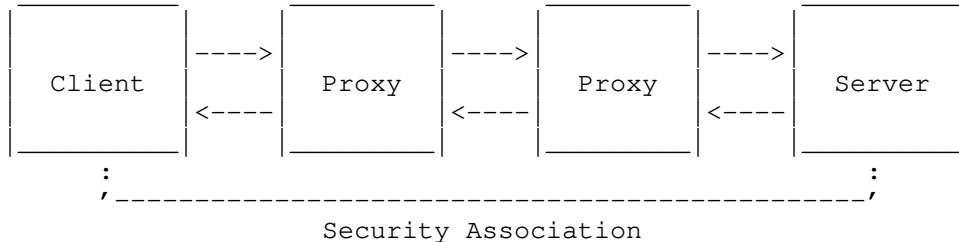


Figure 2: End-to-End Security

This document analyses security aspects of sensor and actuator communications over CoAP that involve proxies (Section 2) and publish-subscribe brokers (Section 3). The analysis is based on the identification of assets associated with these communications and considering the potential threats posed by proxies to these assets. The threat analysis provides the basis for deriving security requirements that a solution for CoAP end-to-end security should meet.

1.1. Assets and Scope

In general, there are the following assets that need to be protected:

- o The devices at the two ends and their (often very constrained) system resources such as available memory, storage, processing power and energy.
- o The physical environment of the devices fitted with sensors and actuators. Access to the physical environment is assumed to be provided through CoAP resources that allow a remote entity to retrieve information about the physical environment (such as the current temperature) or to produce an effect on the physical environment (such as the activation of a heater).
- o The communication infrastructure linking the two devices, which often contains some very constrained networks.
- o The data generated and stored in the involved devices.

An intermediary can directly interfere with the interactions between the two ends and thereby have an impact on all these assets. For example, flooding a device with messages has an impact on system resources, and the successful manipulation of an actuator command

(data generated by an involved device) can have a severe impact on the physical environment. An intermediary can also affect the communication infrastructure, e.g., by dropping messages.

Even if an intermediary is trustworthy, it may be an attractive target for an attack, since such nodes are aggregation points for message flows and may be an easier target from the Internet than the sensor and actuator nodes residing behind them. An intermediary may become subject to intrusion or be infected by malware and perform the attacks of a man-in-the-middle.

The focus of this document is on threats from intermediaries to interactions between two CoAP endpoints. Other types of threats, for example, attacks involving physical access to the CoAP-speaking devices, are out of scope of this document.

Since intermediaries may perform a service for the interacting endpoints, there is a trade-off between the intermediaries' desired functionality and the ability to mitigate threats to the endpoints executed through an intermediary.

1.2. Terminology

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The key word "NOT REQUIRED" is interpreted as synonymous with the key word "OPTIONAL".

2. Proxying

To assess what impact various threats have to the assets, we need to specify and analyse how the proxies operate.

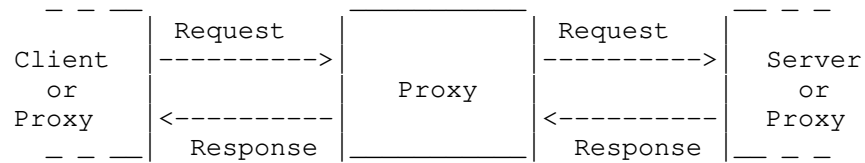


Figure 3: A Proxy

Generally speaking, the functionality of a proxy is to receive a request from a client and to send a response back to that client. There are two ways for the proxy to satisfy the request:

- o The proxy constructs and sends a request to the server indicated in the client's request, receives a response from that server and uses the received data to construct the response to the client.
- o The proxy uses cached data to construct the response to the client.

In both cases, the proxy needs to read some parts both of the request from the client and the response from the server to accomplish its task.

The following subsections analyse the threats posed by a proxy from the perspective of the client on the one hand (Section 2.1.1) and the perspective of the server on the other hand (Section 2.1.2). Section 2.2 then presents the design space for possible security solutions to mitigate the threats.

2.1. Threats and Security Requirements

2.1.1. Client-side

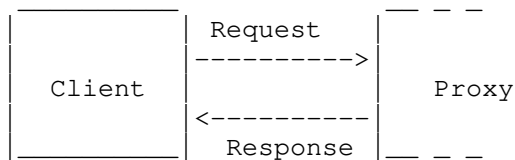


Figure 4: The Client End

The client sends a request to the proxy and waits for a response.

From the perspective of the client, there are three possible flows:

- o The client receives a response.
Reasons include:
 - * The proxy duly processed the request and returns a response based on data it obtained from the origin server.
 - * The proxy encountered an unexpected condition and returns an error response according to specification (e.g., 5.02 Bad Gateway or 5.04 Gateway Timeout).
 - * (Threat 1:) The proxy spoofs a response. For example, the proxy could return a stale or outdated response based on data it previously obtained from the server or some fourth party, or could craft an illicit response itself.
 - * (Threat 2:) The proxy duly processed the request but delays the return of the response.
- o The client does not receive a response.
Reasons include:
 - * The client times out too early.
 - * (Threat 3:) The proxy withholds the response.
- o The client receives too many responses.
Reasons include:
 - * (Threat 4:) The proxy floods the client with responses.

Furthermore, there are threats related to privacy:

- o (Threat 5:) The proxy eavesdrops on the data in the request from the client.
- o (Threat 6:) The proxy measures the size, frequency or distribution of requests from the client.

Note that "cache poisoning" -- the case of caching injected incorrect responses -- is covered from the point of view of the client: it may result in the client receiving a spoofed message or being flooded, or affect other nodes such that the client times out too early.

2.1.1.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the client MUST verify that the response is an "authentic response" before processing it.

The definition of an "authentic response" depends on the desired proxy functionality and protection level (see Section 2.2), but usually means that the client can obtain proof for some or all of the following items:

- o that the requested action was executed by the origin server;
- o that the data originates from the origin server and has not been altered on the way;
- o that the data matches the specifications of the request (such as the target resource);
- o that the data is fresh (when the data is cacheable);
- o that the data is in sequence (when observing a resource).

The proof can, for example, involve a message authentication code that the proxy obtains from the origin server and includes in the response or an additional challenge-response roundtrip.

Exception: A CoAP proxy is specified to return an error response (such as 5.02 Bad Gateway or 5.04 Gateway Timeout) when it encounters an error condition. Since the condition occurs at the proxy and not at the origin server, the response will not be an "authentic response" according to the above definition. (A proxy cannot obtain a proof that the server is unreachable from an unreachable server.) Thus, a client cannot tell if the proxy sends the response according to specification or if it spoofs the response. This threat is NOT REQUIRED to be mitigated by the security solution.

2.1.1.2. Threat 2: Delaying

This threat is REQUIRED to be mitigated by the security solution. Delay attacks are important to mitigate in certain applications, e.g., when using CoAP with actuators. A detailed problem statement and candidate solution can be found in [I-D.mattsson-core-coap-actuators].

2.1.1.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a client cannot tell if the proxy does not send a response because it is hasn't received a response from the origin server yet or if it intentionally withholds the response.

2.1.1.4. Threat 4: Flooding

A CoAP client is specified to reject any response that it does not expect. This can happen before the client verifies whether the response is authentic. Therefore, a flood of responses is primarily a threat to the system resources of the client, in particular to its energy. This threat is NOT REQUIRED to be mitigated by the security solution, but a client SHOULD generally defend against flooding attacks.

2.1.1.5. Threat 5: Eavesdropping

This threat is REQUIRED to be mitigated by the security solution: clients MUST confidentiality protect the data in the requests they send.

Note that this requirement is in conflict with the requirement that the proxy needs to be able to read some parts of the requests in order to accomplish its task. Section 2.2 analyses which parts can be encrypted depending on the desired proxy functionality and protection level. In general, a security solution SHOULD confidentiality protect all data that is not needed by the proxy to accomplish its task.

The keys used for confidentiality protection MUST provide forward secrecy.

2.1.1.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the

feasibility to protect against various threats, e.g., by obfuscating parameters transported in plain text, aligning message flow and traffic between the different cases, adding padding so different messages become indistinguishable, etc.

2.1.2. Server-side

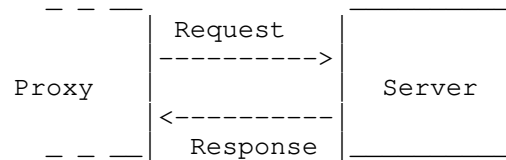


Figure 5: The Server End

A server listens for a request and returns a response.

From the perspective of the server, there are three possible flows:

- o The server receives a request.
Reasons include:
 - * The proxy makes a request on behalf of a client according to specification.
 - * The proxy makes a request (e.g., to validate cached data) on its own behalf.
 - * (Threat 1:) The proxy spoofs a request.
 - * (Threat 2:) The proxy sends a request with delay.
- o The server does not receive a request.
Reasons include:
 - * The proxy does not need to send a request right now.
 - * (Threat 3:) The proxy withholds a request.
- o The server receives too many requests.
Reasons include:
 - * (Threat 4:) The proxy floods the server with requests.

A proxy eavesdropping or inferring information from messages it operates on has an impact on a server in the same way as on a client:

- o (Threat 5:) The proxy eavesdrops on the data in the response from the server.
- o (Threat 6:) The proxy measures the size, frequency or distribution of responses from the server.

2.1.2.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the server MUST verify that the request is an "authentic request" before processing it.

The definition of an "authentic request" depends on the desired proxy functionality and protection level (Section 2.2), but usually means that the server can obtain proof for some or all of the following items:

- o that the proxy acts on behalf of a client;
- o that the data originates from the client and has not been altered on the way;
- o that the request has not been received previously.

The proof can, for example, involve a message authentication code that the proxy obtains from the client and includes in the request or a challenge-response roundtrip.

Exception: A CoAP proxy may make certain requests without acting on behalf of a client (e.g., to validate cached data). Since such a request does not originate from a client, the server cannot tell if the proxy sends the request according to specification or if it spoofs the request. It is up to the security solution how this issue is addressed.

2.1.2.2. Threat 2: Delaying

This threat is REQUIRED to be mitigated by the security solution; see Section 2.1.1.2.

2.1.2.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a server cannot tell if the proxy does not send a request because it has no work to do or if it intentionally withholds a request.

2.1.2.4. Threat 4: Flooding

This threat is NOT REQUIRED to be mitigated by the security solution in particular, but a server SHOULD generally defend against flooding attacks.

2.1.2.5. Threat 5: Eavesdropping

This threat is REQUIRED to be mitigated by the security solution; see Section 2.1.1.5.

2.1.2.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution; see Section 2.1.1.6.

2.2. Solutions

A security solution has to find a trade-off between desired proxy functionality (such as caching) and the provided level of protection. From this trade-off results the definition of what constitutes an "authentic request" or "authentic response" and when a request or response is considered confidentiality protected.

This section presents two exemplary choices of trade-offs:

- o The first case focuses on a high protection level by tying requests and responses uniquely together and confidentiality protecting as much as possible, at the cost of reduced proxy functionality.
- o The second case aims to preserve proxy functionality as much as possible, at the cost of reduced confidentiality protection.

For both cases, this section presents an overview of the functionality and processing rules of the proxy and analyses the required authenticity and confidentiality properties of requests and responses. Due to space constraints, the analysis is limited to the CoAP header, the request and response options shown in Table 1, and the payload.

Requests	Responses
Accept	Content-Format
Content-Format	ETag
ETag	Location-Path
If-Match	Location-Query
If-None-Match	Max-Age
Observe	Observe
Proxy-Scheme	
Proxy-Uri	
Uri-Host	
Uri-Port	
Uri-Path	
Uri-Query	

Table 1: Analysed CoAP Options

Note that, since CoAP was not designed with end-to-end security in mind, a security solution extends the applicability of CoAP beyond its original scope.

2.2.1. Forwarding

In this case we study the functionality of a CoAP forward proxy and assume that caching is disabled. This is applicable to many security critical use cases where a response needs to be securely linked to a unique request from a client and cannot be re-used with another request.

There may be a unique response for each request (see Figure 6) or multiple responses for each request (see Figure 7).

2.2.1.1. Examples

Examples of the need for unique response for each request include alarm status retrieval and actuator command confirmation.

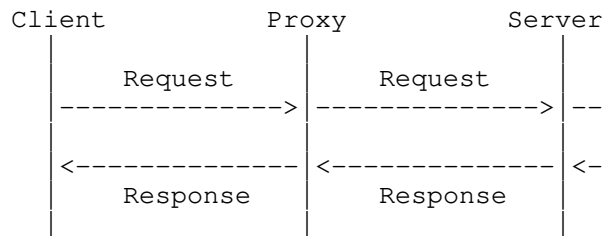


Figure 6: Message Flow with a Unique Response for Each Request

Example: Alarm status retrieval

Figure 6 can be seen as an illustration of a message exchange for a client requesting the alarm status (e.g., GET /alarm_status) from a server. Since the client wants to ensure that the alarm status received is reflecting the current alarm status and not a cached or spoofed response to the same resource, it must be able to verify that the received response is a response to this particular request made by the client. Therefore, the response must be securely linked to the request.

Example: Actuation confirmation

Another example for which Figure 6 serves as illustration is the confirmation of an actuator request. In this case a client, say in an industrial control system, requests a server that a valve should be turned to a certain level, e.g. PUT /valve_42/level with payload "3". In order for the client to correctly evaluate the result of a potential changed valve level, it is important that the client gets a confirmation how the server responded to the requested change, e.g., whether the request was performed or

not. Again, the client wants to ensure that the response is reflecting the result of this particular actuation request made by the client and not a cached or spoofed response. Therefore, the response must be securely linked to the request.

An example of the use of multiple responses for each request is in security critical monitoring scenarios where time synchronization cannot be guaranteed. By avoiding repeated requests from the same client to the same resource, constrained node resources and bandwidth is saved.

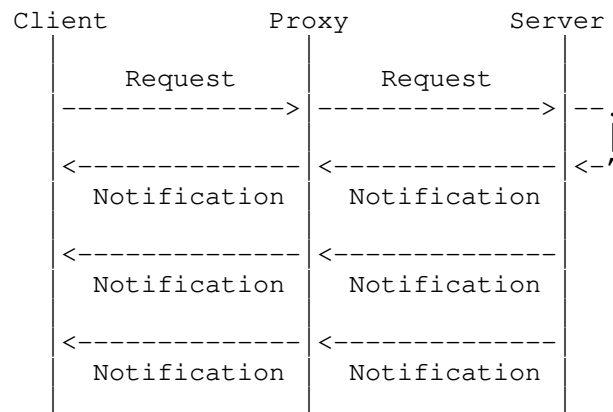


Figure 7: Message Flow of Notifications of Linked to a Unique Request

Example: Secure parameter monitoring

Figure 7 can be seen as an illustration of a message exchange for a client monitoring an important parameter measured by the server, e.g., in a medical or process industry application. The client makes a subscription request for a resource and the server responds with notifications, e.g. providing updates to the parameter on regular time intervals.

The client wants to ensure that the first received notification reflects the current parameter value and that subsequent notifications are timely updates of the initial request. Since notifications may be lost or reordered, the client needs to be able to verify the order of the messages, as sent by the server. By monitoring the received messages and the time they are received, the client can detect missing notifications and take appropriate action.

2.2.1.2. Functional Requirements

- FR1.1 The caching functionality MUST be inhibited; the CoAP option Max-Age of the responses SHALL be 0 (see Section 5.7.1 of [RFC7252]).
- FR1.2 To limit information leaking about the resource (see Section 2.2.1.5) the Proxy-Uri does not contain Uri-Path or Uri-Query.

2.2.1.3. Processing Rules

In this case, the desired proxy functionality is to forward a translated request to the determined destination. There are two modes of operation for requests: Either using the Proxy-Uri option (PR1.1) or using the Proxy-Scheme option together with the Uri-Host, Uri-Port, Uri-Path and Uri-Query options (PR1.2).

- PR1.1 The Proxy-Uri option contains the request URI including request scheme (e.g. "coaps://"); the Proxy-Scheme and Uri-* options are not present.

If the proxy is configured to forward requests to another proxy, then it keeps the Proxy-Uri option; otherwise, it splits the option into its components, adds the corresponding Uri-* options and removes the Proxy-Uri option. Then it makes the request using the request scheme indicated in the Proxy-Uri.

- PR1.2 The Proxy-Scheme option and the Uri-* options together contain the request URI; the Proxy-Uri option is not present.

If the proxy is configured to forward requests to another forwarding proxy, then it keeps the Proxy-Scheme and Uri-* options; otherwise, it removes the Proxy-Scheme option. Then it makes the request using the request scheme indicated in the removed Proxy-Scheme option.

- PR1.3 Responses are forwarded by the proxy, without any modification.

2.2.1.4. Authenticity

A request is considered authentic by the server (Section 2.1.2.1) if the server can obtain proof for all of the following items:

- A1.1 that the proxy acts on behalf of a client;

A1.2 that the following parts of the request originate from the client and have not been altered on the way:

- * the CoAP version,
- * the request method,
- * all options except Proxy-Uri, Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path and Uri-Query, and
- * the payload, if any.

A1.3 that the effective request URI originates from the client and has not been altered on the way;

A1.4 that the request has not been received previously;

A1.5 that the request from the client to the proxy was sent recently.

A response is considered authentic by the client (Section 2.1.1.1) if the client can obtain proof for all of the following items:

A1.6 that the following parts of the response originate from the server and have not been altered on the way:

- * the CoAP version,
- * the response code,
- * all options, and
- * the payload, if any.

A1.7 that the response corresponds uniquely to the request sent by the client.

A1.8 that the response has not been received previously;

A1.9 that the response from the server to the proxy was sent recently;

A1.10 that the response is in sequence if there are multiple responses.

2.2.1.5. Confidentiality

The following parts of the message are confidentiality protected (Section 2.1.1.5):

- o all options except Proxy-Uri, Proxy-Scheme, Uri-Host and Uri-Port;
- o the payload, if any.

2.2.2. Caching

In this case we study caching: how a proxy may serve the same cached response to multiple clients requesting the same resource.

The caching functionality protects communication-constrained servers from repeated requests for the same resources, possibly originating from different clients. This saves system resources, bandwidth, and round-trip time.

There may be one response for each request (see Figure 8) or multiple responses for each request (see Figure 9).

2.2.2.1. Examples

The first example is a simple case of caching.

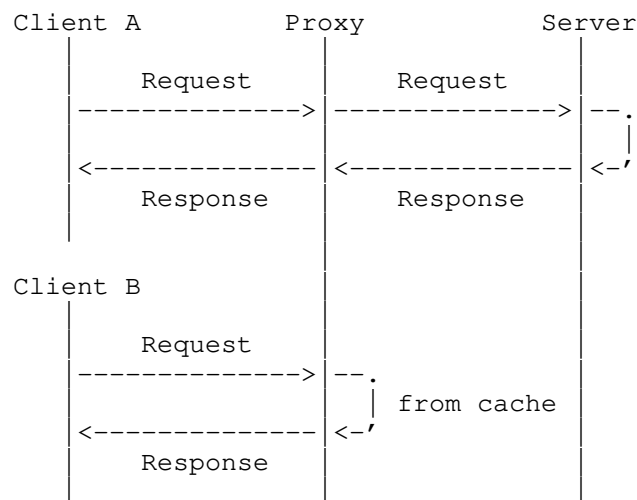


Figure 8: Message Flow for Cached Responses

Example: Caching

In Figure 8, client A requests the proxy to make a certain request to the server and to return the server's response. The proxy services the request by making a request message to the server according to the processing rules. If the server returns a cacheable response, then the proxy stores the response in its cache, performs any necessary translations, and forwards it to the client. Later, client B makes an equivalent request to the proxy that the proxy services by returning the response from its cache. Both client A and B want to verify that the response is valid.

In addition to multiple clients' requests being served by one response, each request may result in multiple responses. The difference compared to Section 2.2.1 is that in this example multiple clients may be served with the same response, further saving server resources.

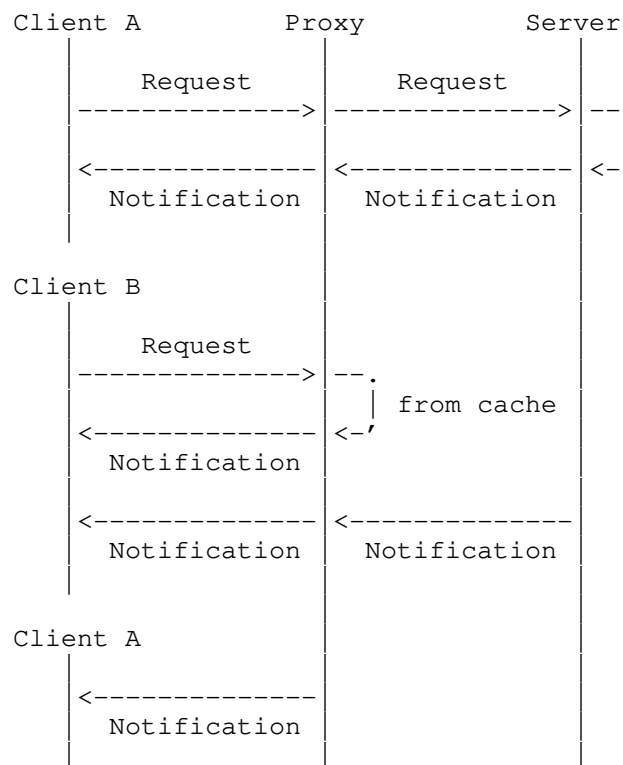


Figure 9: Message Flow for Observe with Multiple Observers

Example: Observe with caching

In Figure 9, the server exposes an observable resource (e.g., the current reading of a temperature sensor). Multiple clients are interested in the current state of the resource and observe it using the CoAP resource observation mechanism [RFC7641]. The goal is to keep the state observed by the clients closely in sync with the actual state of the resource at the server. Another goal is to minimize the burden on the server by moving the task to fan out notifications to multiple clients from the server to the proxy.

2.2.2.2. Functional Requirements

The security solution SHOULD protect requests and responses in a way that a proxy can perform the following tasks:

- FR2.1 Storing a cacheable response in a cache. This requires that the proxy is able to calculate the cache-key of the request. Cacheable responses include 2.05 (Content) responses and all error responses.
- FR2.2 Returning a fresh response from its cache without contacting the server.
- FR2.3 Performing validation of a response cached by the proxy as well as validation of a response cached by the client.
- FR2.4 Observing a resource on behalf of one or more clients.

2.2.2.3. Processing Rules

The proxy complies with the forwarding rules PR1.1 - 1.3 (Section 2.2.1.3) and the rules below. The rules below have priority.

- PR2.1 If the proxy receives a request where the cache key matches that of a cached fresh response, then the proxy with that response; otherwise, it makes a request towards the server.
- PR2.2 The proxy caches and forwards cacheable responses. If there is already a response in the cache with the cache key of the corresponding request, then the old response in the cache is marked as stale.
- PR2.3 If the proxy receives a request that contains an ETag option and the proxy has a fresh response with the same cache key and ETag, then the proxy replies to the request with a 2.03 (Valid) response without payload, else it forwards a translated request.

PR2.4 The proxy updates the Max-Age option according to the Max-Age associated with the resource representation it receives, decreasing its value to reflect the time spent in the cache.

PR2.5 If the request contains an Accept option and if there is a fresh response that matches the cache key for the corresponding request except for the Accept option and if the Content-Format of the response matches that of the Accept option, then the proxy forwards the cached response to the requesting client.

2.2.2.4. Authenticity

A request is considered authentic by the server (Section 2.1.2.1) if the server can obtain proof for all of the following items:

A2.1 that the following parts of the request originate from the client and have not been altered on the way:

- * the CoAP version,
- * the request method,
- * all options except ETag, Observe, Proxy-Uri, Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path and Uri-Query, and
- * the payload, if any.

A2.2 that the effective request URI originates from the client and has not been altered on the way;

A response is considered authentic by the client (Section 2.1.1.1) if the client can obtain proof for all of the following items:

A2.3 that the following parts of the response originate from the server and have not been altered on the way:

- * the CoAP version,
- * the response code,
- * all options except Max-Age and Observe, and
- * the payload, if any.

A2.4 that the response matches the specifications of the request;

A2.5 that the data is fresh (when the response is cacheable);

A2.6 that the response is in sequence (when observing a resource).

2.2.2.5. Confidentiality

No parts of a request are confidentiality protected
(Section 2.1.2.5).

A response is considered confidentiality protected (Section 2.1.2.5)
if the payload of the response is confidentiality protected.

3. Publish-Subscribe

Much of the concerns about proxies as described previously in this document also applies to other kinds of intermediary nodes. In this section we study brokers in a publish-subscribe setting [I-D.ietf-core-coap-pubsub]. The case of combining brokers and proxies is out of scope for this version of the document.

There are different ways for a pub-sub broker to operate. We consider the following broker operations:

- o The broker receives a request for a topic from a subscriber.
- o The broker receives a request for a publication to a topic from a publisher and forwards the publication to the subscribers of the topic.

We consider the setting where there is a security association between publisher and subscriber such that the publications can be protected during transfer, see Figure 10.

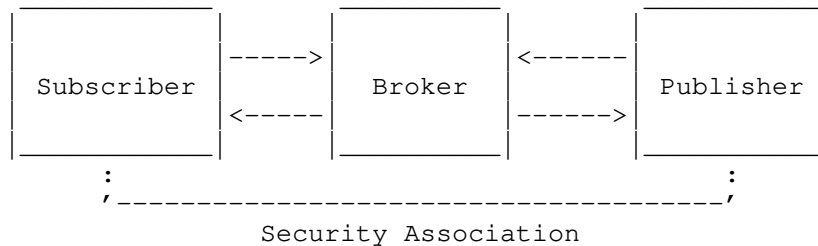


Figure 10: Publisher-to-Subscriber Security

Since there is no security association with the broker, we only consider the subscribe and publish functionality of the broker. Note that the broker needs to read the topic to accomplish this task.

3.1. Threats and Security Requirements

3.1.1. Subscriber-side

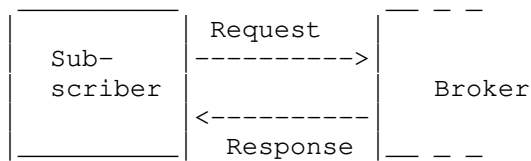


Figure 11: The Subscriber End

The subscriber sends a subscription request to the broker and waits for a response.

From the perspective of the subscriber, there are three possible flows:

- o The subscriber receives a response.
Reasons include:
 - * The broker duly processed the request and returns a response based on data it obtained from a publisher.
 - * The subscriber made a bad request and the broker returns an error response accordingly (e.g., 4.04 Not Found).
 - * The broker encountered an unexpected condition and returns an error response accordingly (e.g., 5.03 Service Unavailable).
 - * (Threat 1:) The broker spoofs a response.
 - * (Threat 2:) The broker duly processed the request but delays the return of a response.
- o The subscriber does not receive a response.
Reasons include:
 - * The subscriber times out too early.
 - * (Threat 3:) The broker withholds a response.
- o The subscriber receives too many responses.
Reasons include:
 - * (Threat 4:) The broker floods the subscriber with responses.

Furthermore, there are threats related to privacy:

- o (Threat 5:) The broker eavesdrops on the data in the request from the subscriber.

- o (Threat 6:) The broker measures the size, frequency or distribution of requests from the subscriber.

Note that "topic poisoning" -- the case of storing injected incorrect publications -- is covered from the point of view of the subscriber: it may result in the subscriber receiving a spoofed message, or being flooded, or affect other nodes such that the subscriber times out too early.

3.1.1.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the subscriber MUST verify that a response is an "authentic publication" before processing it.

The definition of an "authentic publication" depends on the setting (Section 3.2), but usually means that the subscriber can obtain proof for some or all of the following items:

- o that the data matches the specifications of the request (such as the topic);
- o that the data originates from a publisher that is authorized to publish to the topic;
- o that the data has not been altered on the way between publisher and subscriber;
- o that the data is fresh (when the data is cacheable);
- o that the data is in sequence (when observing a topic).

The proof can, for example, include a message authentication code that the proxy obtains from the origin server and includes in the response or an additional challenge-response roundtrip.

Exception: A CoAP server like the broker is specified to return an error response (such as 4.04 Not Found or 5.03 Service Unavailable) when it encounters an error condition. Since the condition occurs at the broker and not at the publisher, the response will not be an "authentic response" according to the above definition. Thus, a subscriber cannot tell if the broker sends the error response according to specification or if it spoofs the response. This threat is NOT REQUIRED to be mitigated by the security solution.

3.1.1.2. Threat 2: Delaying

This threat is NOT REQUIRED to be mitigated by the security solution.

3.1.1.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a subscriber cannot tell if the broker does not send a response because it is hasn't received a publication from the publisher yet or if it intentionally withholds the response.

3.1.1.4. Threat 4: Flooding

A CoAP client like the subscriber is specified to reject any response that it does not expect. This can happen before the subscriber verifies if the response is authentic. Therefore, a flood of responses is primarily a threat to the system resources of the client, in particular to its energy. This threat is NOT REQUIRED to be mitigated by the security solution, but a subscriber SHOULD generally defend against flooding attacks.

3.1.1.5. Threat 5: Eavesdropping

This threat is NOT REQUIRED to be mitigated: The broker needs to read all parts of the request from the subscriber to accomplish its task.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the feasibility to protect against various threats, e.g., by obfuscating topic content.

3.1.1.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the feasibility to protect against various threats, e.g., by obfuscating parameters transported in plain text, aligning message flow and traffic between the different cases, adding padding so different messages become indistinguishable, etc.

3.1.2. Publisher-side

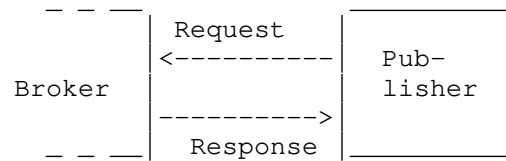


Figure 12: The Publisher End

The publisher sends a publication request to the broker and waits for a response.

The threat of the broker eavesdropping on the data in the publication request is REQUIRED to be mitigated by the security solution: publishers MUST confidentiality protect the data in the requests they send. This excludes parts that the broker needs to read to perform its job, e.g., the topic.

The threat of the broker measuring the size, frequency or distribution of publication requests is NOT REQUIRED to be mitigated by the security solution; see Section 3.1.1.6.

The broker is in full control of the response and may therefore arbitrarily spoof, delay, or withhold it. This threat is NOT REQUIRED to be mitigated. For example, a proof that the broker has notified all subscribers is NOT REQUIRED.

3.2. Solutions

3.2.1. Brokering

In this case we study brokering: how a broker may serve the same publication to multiple subscribers observing the same topic.

The brokering functionality protects communication-constrained publishers from repeated requests for the same resources, possibly originating from different subscribers. This saves system resources, bandwidth, and round-trip time.

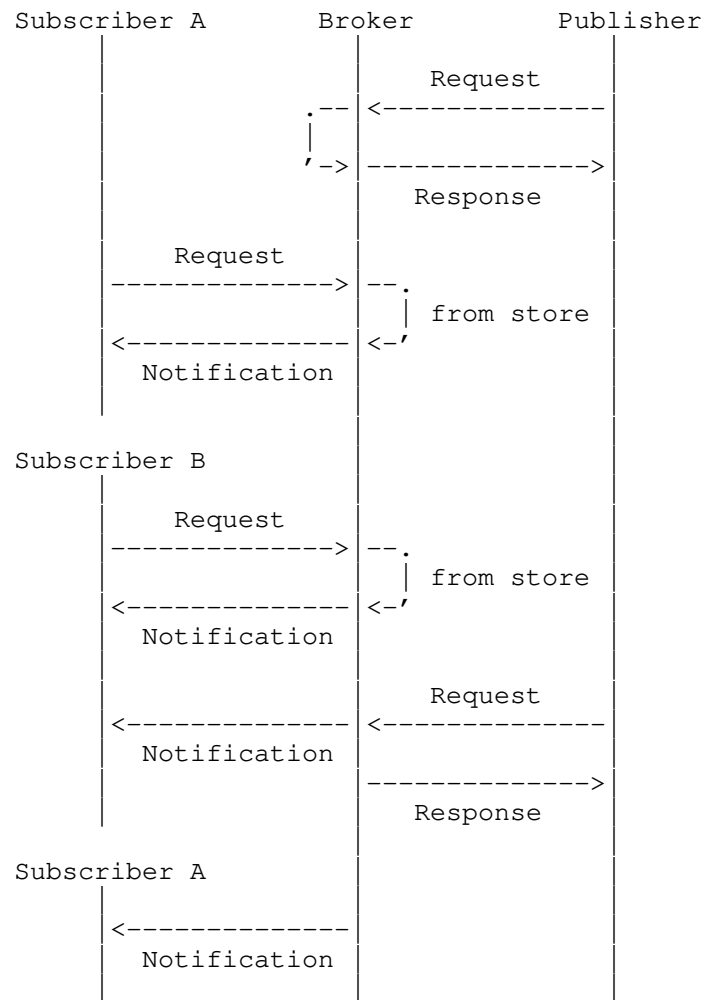


Figure 13: Message Flow for Publish Subscribe

Example

In Figure 13, the publisher publishes to a topic (e.g., the current reading of a temperature sensor). Multiple subscribers are interested in the current state of the topic and observe the topic as specified in [I-D.ietf-core-coap-pubsub]. The goal is to keep the state observed by the subscribers closely in sync with the actual state of the resource at the publisher. Another goal is to minimize the burden on the publisher by moving the task to fan out notifications to multiple subscribers from the publisher to the broker.

3.2.1.1. Functional Requirements

The security solution SHOULD protect subscription and publication requests in a way that a broker can perform the following tasks:

FR3.1 Storing publications. This requires that the broker is able to read the topic of the request.

FR3.2 Returning a stored publication without contacting the publisher.

3.2.1.2. Processing Rules

The broker complies with the following rules:

PR3.1 If the broker receives a request where the topic matches that of a cached publication, then the broker responds with that publication.

PR3.2 The broker caches and forwards publication notifications.

3.2.1.3. Authenticity

A publication is considered authentic by the subscriber if the subscriber can obtain proof for all all of the following items:

A3.1 that the payload is associated to the topic;

A3.2 that the payload has not been altered since published;

A3.3 that the publication is in sequence.

3.2.1.4. Confidentiality

The payload of a publication request is confidentiality protected.

4. Security Considerations

This document is about security; as such, there are no additional security considerations.

5. IANA Considerations

This document includes no request to IANA.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

6.2. Informative References

- [I-D.ietf-core-coap-pubsub] Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-02 (work in progress), July 2017.
- [I-D.mattsson-core-coap-actuators] Mattsson, J., Fornehed, J., Selander, G., and F. Palombini, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-02 (work in progress), November 2016.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<http://www.rfc-editor.org/info/rfc8152>>.

Acknowledgments

Thanks to Ari Keranen, John Mattsson, Jim Schaad and Ludwig Seitz for helpful comments and discussions that have shaped the document.

Authors' Addresses

Goeran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen 28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

C. Bormann
Universitaet Bremen TZI
Z. Shelby, Ed.
ARM
March 21, 2016

Block-wise transfers in CoAP
draft-ietf-core-block-19

Abstract

CoAP is a RESTful transfer protocol for constrained nodes and networks. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications will need to transfer larger payloads -- for instance, for firmware updates. With HTTP, TCP does the grunt work of slicing large payloads up into multiple packets and ensuring that they all arrive and are handled in the right order.

CoAP is based on datagram transports such as UDP or DTLS, which limits the maximum size of resource representations that can be transferred without too much fragmentation. Although UDP supports larger payloads through IP fragmentation, it is limited to 64 KiB and, more importantly, doesn't really work well for constrained applications and networks.

Instead of relying on IP fragmentation, this specification extends basic CoAP with a pair of "Block" options, for transferring multiple blocks of information from a resource representation in multiple request-response pairs. In many important cases, the Block options enable a server to be truly stateless: the server can handle each block transfer separately, with no need for a connection setup or other server-side memory of previous block transfers.

In summary, the Block options provide a minimal way to transfer larger representations in a block-wise fashion.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Block-wise transfers	5
2.1. The Block2 and Block1 Options	5
2.2. Structure of a Block Option	6
2.3. Block Options in Requests and Responses	8
2.4. Using the Block2 Option	10
2.5. Using the Block1 Option	12
2.6. Combining Block-wise Transfers with the Observe Option .	13
2.7. Combining Block1 and Block2	14
2.8. Combining Block2 with Multicast	14
2.9. Response Codes	14
2.9.1. 2.31 Continue	15
2.9.2. 4.08 Request Entity Incomplete	15
2.9.3. 4.13 Request Entity Too Large	15
2.10. Caching Considerations	15
3. Examples	16
3.1. Block2 Examples	16
3.2. Block1 Examples	20
3.3. Combining Block1 and Block2	21
3.4. Combining Observe and Block2	23
4. The Size2 and Size1 Options	26
5. HTTP Mapping Considerations	28
6. IANA Considerations	29
7. Security Considerations	29

7.1. Mitigating Resource Exhaustion Attacks	30
7.2. Mitigating Amplification Attacks	31
8. Acknowledgements	31
9. References	32
9.1. Normative References	32
9.2. Informative References	32
Authors' Addresses	33

1. Introduction

The work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes (such as microcontrollers with limited RAM and ROM [RFC7228]) and networks (such as 6LoWPAN, [RFC4944]) [RFC7252]. The CoAP protocol is intended to provide RESTful [REST] services not unlike HTTP [RFC7230], while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

This objective requires restraint in a number of sometimes conflicting ways:

- o reducing implementation complexity in order to minimize code size,
- o reducing message sizes in order to minimize the number of fragments needed for each message (in turn to maximize the probability of delivery of the message), the amount of transmission power needed and the loading of the limited-bandwidth channel,
- o reducing requirements on the environment such as stable storage, good sources of randomness or user interaction capabilities.

CoAP is based on datagram transports such as UDP, which limit the maximum size of resource representations that can be transferred without creating unreasonable levels of IP fragmentation. In addition, not all resource representations will fit into a single link layer packet of a constrained network, which may cause adaptation layer fragmentation even if IP layer fragmentation is not required. Using fragmentation (either at the adaptation layer or at the IP layer) for the transport of larger representations would be possible up to the maximum size of the underlying datagram protocol (such as UDP), but the fragmentation/reassembly process burdens the lower layers with conversation state that is better managed in the application layer.

The present specification defines a pair of CoAP options to enable block-wise access to resource representations. The Block options provide a minimal way to transfer larger resource representations in a block-wise fashion. The overriding objective is to avoid the need for creating conversation state at the server for block-wise GET requests. (It is impossible to fully avoid creating conversation state for POST/PUT, if the creation/replacement of resources is to be atomic; where that property is not needed, there is no need to create server conversation state in this case, either.)

In summary, this specification adds a pair of Block options to CoAP that can be used for block-wise transfers. Benefits of using these options include:

- o Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- o No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- o The transfer of each block is acknowledged, enabling individual retransmission if required.
- o Both sides have a say in the block size that actually will be used.
- o The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- o If needed, the Block options can also be used (without changes) to provide random access to power-of-two sized blocks within a resource representation.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119] and indicate requirement levels for compliant CoAP implementations.

In this document, the term "byte" is used in its now customary sense as a synonym for "octet".

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C, except that the operator "***" stands for exponentiation.

2. Block-wise transfers

As discussed in the introduction, there are good reasons to limit the size of datagrams in constrained networks:

- o by the maximum datagram size (~ 64 KiB for UDP)
- o by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- o by the desire to avoid adaptation layer fragmentation (60-80 bytes for 6LoWPAN [RFC4919])

When a resource representation is larger than can be comfortably transferred in the payload of a single CoAP datagram, a Block option can be used to indicate a block-wise transfer. As payloads can be sent both with requests and with responses, this specification provides two separate options for each direction of payload transfer. In naming these options (for block-wise transfers as well as in Section 4), we use the number 1 ("Block1", "Size1") to refer to the transfer of the resource representation that pertains to the request, and the number 2 ("Block2", "Size2") to refer to the transfer of the resource representation for the response.

In the following, the term "payload" will be used for the actual content of a single CoAP message, i.e. a single block being transferred, while the term "body" will be used for the entire resource representation that is being transferred in a block-wise fashion. The Content-Format option applies to the body, not to the payload, in particular the boundaries between the blocks may be in places that are not separating whole units in terms of the structure, encoding, or content-coding used by the Content-Format.

In most cases, all blocks being transferred for a body (except for the last one) will be of the same size. The block size is not fixed by the protocol. To keep the implementation as simple as possible, the Block options support only a small range of power-of-two block sizes, from 2^4 (16) to 2^{10} (1024) bytes. As bodies often will not evenly divide into the power-of-two block size chosen, the size need not be reached in the final block (but even for the final block, the chosen power-of-two size will still be indicated in the block size field of the Block option).

2.1. The Block2 and Block1 Options

No.	C	U	N	R	Name	Format	Length	Default
23	C	U	-	-	Block2	uint	0-3	(none)
27	C	U	-	-	Block1	uint	0-3	(none)

Table 1: Block Option Numbers

Both Block1 and Block2 options can be present both in request and response messages. In either case, the Block1 Option pertains to the request payload, and the Block2 Option pertains to the response payload.

Hence, for the methods defined in [RFC7252], Block1 is useful with the payload-bearing POST and PUT requests and their responses. Block2 is useful with GET, POST, and PUT requests and their payload-bearing responses (2.01, 2.02, 2.04, 2.05 -- see section "Payload" of [RFC7252]).

Where Block1 is present in a request or Block2 in a response (i.e., in that message to the payload of which it pertains) it indicates a block-wise transfer and describes how this specific block-wise payload forms part of the entire body being transferred ("descriptive usage"). Where it is present in the opposite direction, it provides additional control on how that payload will be formed or was processed ("control usage").

Implementation of either Block option is intended to be optional. However, when it is present in a CoAP message, it MUST be processed (or the message rejected); therefore it is identified as a critical option. It MUST NOT occur more than once.

2.2. Structure of a Block Option

Three items of information may need to be transferred in a Block (Block1 or Block2) option:

- o The size of the block (SZX);
- o whether more blocks are following (M);
- o the relative number of the block (NUM) within a sequence of blocks with the given size.

The value of the Block Option is a variable-size (0 to 3 byte) unsigned integer (uint, see Section 3.2 of [RFC7252]). This integer

value encodes these three fields, see Figure 1. (Due to the CoAP uint encoding rules, when all of NUM, M, and SZX happen to be zero, a zero-byte integer will be sent.)

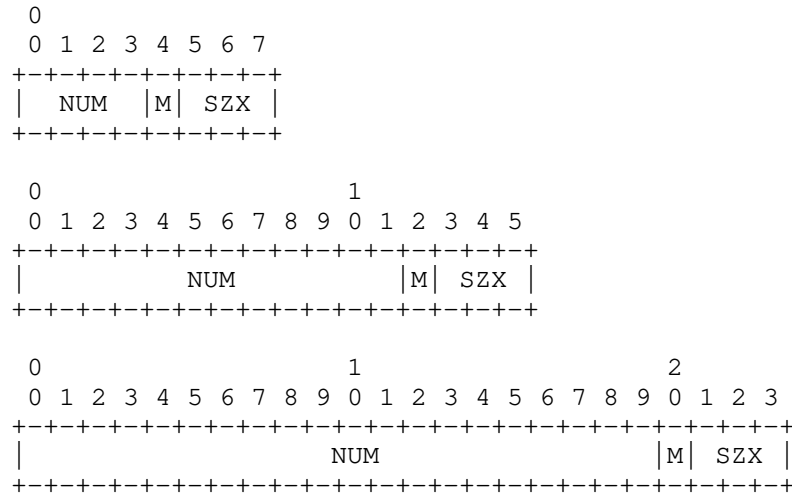


Figure 1: Block option value

The block size is encoded using a three-bit unsigned integer (0 for 2^{**4} to 6 for 2^{**10} bytes), which we call the "SZX" ("size exponent"); the actual block size is then $2^{**}(\text{SZX} + 4)$. SZX is transferred in the three least significant bits of the option value (i.e., $\text{val} \& 7$ where "val" is the value of the option).

The fourth least significant bit, the M or "more" bit ($\text{val} \& 8$), indicates whether more blocks are following or the current block-wise transfer is the last block being transferred.

The option value divided by sixteen (the NUM field) is the sequence number of the block currently being transferred, starting from zero. The current transfer is therefore about the "size" bytes starting at byte $\text{NUM} \ll (\text{SZX} + 4)$.

Implementation note: As an implementation convenience, $(\text{val} \& \sim 0xF) \ll (\text{val} \& 7)$, i.e., the option value with the last 4 bits masked out, shifted to the left by the value of SZX, gives the byte position of the first byte of the block being transferred.

More specifically, within the option value of a Block1 or Block2 Option, the meaning of the option fields is defined as follows:

NUM: Block Number, indicating the block number being requested or provided. Block number 0 indicates the first block of a body (i.e., starting with the first byte of the body).

M: More Flag ("not last block"). For descriptive usage, this flag, if unset, indicates that the payload in this message is the last block in the body; when set it indicates that there are one or more additional blocks available. When a Block2 Option is used in a request to retrieve a specific block number ("control usage"), the M bit MUST be sent as zero and ignored on reception. (In a Block1 Option in a response, the M flag is used to indicate atomicity, see below.)

SZX: Block Size. The block size is represented as three-bit unsigned integer indicating the size of a block to the power of two. Thus block size = $2^{(SZX + 4)}$. The allowed values of SZX are 0 to 6, i.e., the minimum block size is $2^{(0+4)} = 16$ and the maximum is $2^{(6+4)} = 1024$. The value 7 for SZX (which would indicate a block size of 2048) is reserved, i.e. MUST NOT be sent and MUST lead to a 4.00 Bad Request response code upon reception in a request.

There is no default value for the Block1 and Block2 Options. Absence of one of these options is equivalent to an option value of 0 with respect to the value of NUM and M that could be given in the option, i.e. it indicates that the current block is the first and only block of the transfer (block number 0, M bit not set). However, in contrast to the explicit value 0, which would indicate an SZX of 0 and thus a size value of 16 bytes, there is no specific explicit size implied by the absence of the option -- the size is left unspecified. (As for any uint, the explicit value 0 is efficiently indicated by a zero-length option; this, therefore, is different in semantics from the absence of the option.)

2.3. Block Options in Requests and Responses

The Block options are used in one of three roles:

- o In descriptive usage, i.e., a Block2 Option in a response (such as a 2.05 response for GET), or a Block1 Option in a request (a PUT or POST):
 - * The NUM field in the option value describes what block number is contained in the payload of this message.
 - * The M bit indicates whether further blocks need to be transferred to complete the transfer of that body.

- * The block size implied by SZX MUST match the size of the payload in bytes, if the M bit is set. (SZX does not govern the payload size if M is unset). For Block2, if the request suggested a larger value of SZX, the next request MUST move SZX down to the size given in the response. (The effect is that, if the server uses the smaller of (1) its preferred block size and (2) the block size requested, all blocks for a body use the same block size.)
- o A Block2 Option in control usage in a request (e.g., GET):
 - * The NUM field in the Block2 Option gives the block number of the payload that is being requested to be returned in the response.
 - * In this case, the M bit has no function and MUST be set to zero.
 - * The block size given (SZX) suggests a block size (in the case of block number 0) or repeats the block size of previous blocks received (in the case of a non-zero block number).
- o A Block1 Option in control usage in a response (e.g., a 2.xx response for a PUT or POST request):
 - * The NUM field of the Block1 Option indicates what block number is being acknowledged.
 - * If the M bit was set in the request, the server can choose whether to act on each block separately, with no memory, or whether to handle the request for the entire body atomically, or any mix of the two.
 - + If the M bit is also set in the response, it indicates that this response does not carry the final response code to the request, i.e. the server collects further blocks from the same endpoint and plans to implement the request atomically (e.g., acts only upon reception of the last block of payload). In this case, the response MUST NOT carry a Block2 option.
 - + Conversely, if the M bit is unset even though it was set in the request, it indicates the block-wise request was enacted now specifically for this block, and the response carries the final response to this request (and to any previous ones with the M bit set in the response's Block1 Option in this sequence of block-wise transfers); the client is still

expected to continue sending further blocks, the request method for which may or may not also be enacted per-block.

- * Finally, the SZX block size given in a control Block1 Option indicates the largest block size preferred by the server for transfers toward the resource that is the same or smaller than the one used in the initial exchange; the client SHOULD use this block size or a smaller one in all further requests in the transfer sequence, even if that means changing the block size (and possibly scaling the block number accordingly) from now on.

Using one or both Block options, a single REST operation can be split into multiple CoAP message exchanges. As specified in [RFC7252], each of these message exchanges uses their own CoAP Message ID.

The Content-Format Option sent with the requests or responses MUST reflect the content-format of the entire body. If blocks of a response body arrive with different content-format options, it is up to the client how to handle this error (it will typically abort any ongoing block-wise transfer). If blocks of a request arrive at a server with mismatching content-format options, the server MUST NOT assemble them into a single request; this usually leads to a 4.08 (Request Entity Incomplete, Section 2.9.2) error response on the mismatching block.

2.4. Using the Block2 Option

When a request is answered with a response carrying a Block2 Option with the M bit set, the requester may retrieve additional blocks of the resource representation by sending further requests with the same options as the initial request and a Block2 Option giving the block number and block size desired. In a request, the client MUST set the M bit of a Block2 Option to zero and the server MUST ignore it on reception.

To influence the block size used in a response, the requester MAY also use the Block2 Option on the initial request, giving the desired size, a block number of zero and an M bit of zero. A server MUST use the block size indicated or a smaller size. Any further block-wise requests for blocks beyond the first one MUST indicate the same block size that was used by the server in the response for the first request that gave a desired size using a Block2 Option.

Once the Block2 Option is used by the requester and a first response has been received with a possibly adjusted block size, all further requests in a single block-wise transfer SHOULD ultimately use the same size, except that there may not be enough content to fill the

last block (the one returned with the M bit not set). (Note that the client may start using the Block2 Option in a second request after a first request without a Block2 Option resulted in a Block2 option in the response.) The server SHOULD use the block size indicated in the request option or a smaller size, but the requester MUST take note of the actual block size used in the response it receives to its initial request and proceed to use it in subsequent requests. The server behavior MUST ensure that this client behavior results in the same block size for all responses in a sequence (except for the last one with the M bit not set, and possibly the first one if the initial request did not contain a Block2 Option).

Block-wise transfers can be used to GET resources the representations of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing resources. In the latter case, the Block2 Option SHOULD be used in conjunction with the ETag Option, to ensure that the blocks being reassembled are from the same version of the representation: The server SHOULD include an ETag option in each response. If an ETag option is available, the client's reassembler, when reassembling the representation from the blocks being exchanged, MUST compare ETag Options. If the ETag Options do not match in a GET transfer, the requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server MAY cache the current value of a representation for an ongoing sequence of requests. (The server may identify the sequence by the combination of the requesting end-point and the URI being the same in each block-wise request.) Note well that this specification makes no requirement for the server to establish any state; however, servers that offer quickly changing resources may thereby make it impossible for a client to ever retrieve a consistent set of blocks. Clients that want to retrieve all blocks of a resource SHOULD strive to do so without undue delay. Servers can fully expect to be free to discard any cached state after a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) after the last access to the state, however, there is no requirement to always keep the state for as long.

The Block2 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise response payload transfer (e.g., GET) operations to the same resource. This is rarely a requirement, but as a workaround, a client may vary the cache key (e.g., by using one of several URIs accessing resources with the same semantics, or by varying a proxy-safe elective option).

2.5. Using the Block1 Option

In a request with a request payload (e.g., PUT or POST), the Block1 Option refers to the payload in the request (descriptive usage).

In response to a request with a payload (e.g., a PUT or POST transfer), the block size given in the Block1 Option indicates the block size preference of the server for this resource (control usage). Obviously, at this point the first block has already been transferred by the client without benefit of this knowledge. Still, the client SHOULD heed the preference indicated and, for all further blocks, use the block size preferred by the server or a smaller one. Note that any reduction in the block size may mean that the second request starts with a block number larger than one, as the first request already transferred multiple blocks as counted in the smaller size.

To counter the effects of adaptation layer fragmentation on packet delivery probability, a client may want to give up retransmitting a request with a relatively large payload even before MAX_RETRANSMIT has been reached, and try restating the request as a block-wise transfer with a smaller payload. Note that this new attempt is then a new message-layer transaction and requires a new Message ID. (Because of the uncertainty whether the request or the acknowledgement was lost, this strategy is useful mostly for idempotent requests.)

In a block-wise transfer of a request payload (e.g., a PUT or POST) that is intended to be implemented in an atomic fashion at the server, the actual creation/replacement takes place at the time the final block, i.e. a block with the M bit unset in the Block1 Option, is received. In this case, all success responses to non-final blocks carry the response code 2.31 (Continue, Section 2.9.1). If not all previous blocks are available at the server at the time of processing the final block, the transfer fails and error code 4.08 (Request Entity Incomplete, Section 2.9.2) MUST be returned. A server MAY also return a 4.08 error code for any (final or non-final) Block1 transfer that is not in sequence; clients that do not have specific mechanisms to handle this case therefore SHOULD always start with block zero and send the following blocks in order.

One reason that a client might encounter a 4.08 error code is that the server has already timed out and discarded the partial request body being assembled. Clients SHOULD strive to send all blocks of a request without undue delay. Servers can fully expect to be free to discard any partial request body when a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) has elapsed after the most recent block

was transferred; however, there is no requirement on a server to always keep the partial request body for as long.

The error code 4.13 (Request Entity Too Large) can be returned at any time by a server that does not currently have the resources to store blocks for a block-wise request payload transfer that it would intend to implement in an atomic fashion. (Note that a 4.13 response to a request that does not employ Block1 is a hint for the client to try sending Block1, and a 4.13 response with a smaller SZX in its Block1 option than requested is a hint to try a smaller SZX.)

The Block1 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations to the same resource. Starting a new block-wise sequence of requests to the same resource (before an old sequence from the same endpoint was finished) simply overwrites the context the server may still be keeping. (This is probably exactly what one wants in this case - the client may simply have restarted and lost its knowledge of the previous sequence.)

2.6. Combining Block-wise Transfers with the Observe Option

The Observe Option provides a way for a client to be notified about changes over time of a resource [RFC7641]. Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. The following rules apply to the combination of block-wise transfers with notifications.

Observation relationships always apply to an entire resource; the Block2 option does not provide a way to observe a single block of a resource.

As with basic GET transfers, the client can indicate its desired block size in a Block2 Option in the GET request establishing or renewing the observation relationship. If the server supports block-wise transfers, it SHOULD take note of the block size and apply it as a maximum size to all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the entry in that list is updated by the server receiving a new GET request for the resource from the client).

When sending a 2.05 (Content) notification, the server only sends the first block of the representation. The client retrieves the rest of the representation as if it had caused this first response by a GET request, i.e., by using additional GET requests with Block2 options containing NUM values greater than zero. (This results in the transfer of the entire representation, even if only some of the blocks have changed with respect to a previous notification.)

As with other dynamically changing resources, to ensure that the blocks being reassembled are from the same version of the representation, the server SHOULD include an ETag option in each response, and the reassembling client MUST compare the ETag options (Section 2.4). Even more so than for the general case of Block2, clients that want to retrieve all blocks of a resource they have been notified about with a first block SHOULD strive to do so without undue delay.

See Section 3.4 for examples.

2.7. Combining Block1 and Block2

In PUT and particularly in POST exchanges, both the request body and the response body may be large enough to require the use of block-wise transfers. First, the Block1 transfer of the request body proceeds as usual. In the exchange of the last slice of this block-wise transfer, the response carries the first slice of the Block2 transfer (NUM is zero). To continue this Block2 transfer, the client continues to send requests similar to the requests in the Block1 phase, but leaves out the Block1 options and includes a Block2 request option with non-zero NUM.

Block2 transfers that retrieve the response body for a request that used Block1 MUST be performed in sequential order.

2.8. Combining Block2 with Multicast

A client can use the Block2 option in a multicast GET request with NUM = 0 to aid in limiting the size of the response.

Similarly, a response to a multicast GET request can use a Block2 option with NUM = 0 if the representation is large, or to further limit the size of the response.

In both cases, the client retrieves any further blocks using unicast exchanges; in the unicast requests, the client SHOULD heed any block size preferences indicated by the server in the response to the multicast request.

Other uses of the Block options in conjunction with multicast messages are for further study.

2.9. Response Codes

Two response codes are defined by this specification beyond those already defined in [RFC7252], and another response code is extended in its meaning.

2.9.1. 2.31 Continue

This new success status code indicates that the transfer of this block of the request body was successful and that the server encourages sending further blocks, but that a final outcome of the whole block-wise request cannot yet be determined. No payload is returned with this response code.

2.9.2. 4.08 Request Entity Incomplete

This new client error status code indicates that the server has not received the blocks of the request body that it needs to proceed. The client has not sent all blocks, not sent them in the order required by the server, or has sent them long enough ago that the server has already discarded them.

2.9.3. 4.13 Request Entity Too Large

In [RFC7252], section 5.9.2.9, the response code 4.13 (Request Entity Too Large) is defined to be like HTTP 413 "Request Entity Too Large". [RFC7252] also recommends that this response SHOULD include a Size1 Option (Section 4) to indicate the maximum size of request entity the server is able and willing to handle, unless the server is not in a position to make this information available.

The present specification allows the server to return this response code at any time during a Block1 transfer to indicate that it does not currently have the resources to store blocks for a transfer that it would intend to implement in an atomic fashion. It also allows the server to return a 4.13 response to a request that does not employ Block1 as a hint for the client to try sending Block1. Finally, a 4.13 response to a request with a Block1 option (control usage, see Section 2.3) where the response carries a smaller SZX in its Block1 option is a hint to try that smaller SZX.

2.10. Caching Considerations

This specification attempts to leave a variety of implementation strategies open for caches, in particular those in caching proxies. E.g., a cache is free to cache blocks individually, but also could wait to obtain the complete representation before it serves parts of it. Partial caching may be more efficient in a cross-proxy (equivalent to a streaming HTTP proxy). A cached block (partial cached response) can be used in place of a complete response to satisfy a block-wise request that is presented to a cache. Note that different blocks can have different Max-Age values, as they are transferred at different times. A response with a block updates the freshness of the complete representation. Individual blocks can be

validated, and validating a single block validates the complete representation. A response with a Block1 Option in control usage with the M bit set invalidates cached responses for the target URI.

A cache or proxy that combines responses (e.g., to split blocks in a request or increase the block size in a response, or a cross-proxy) may need to combine 2.31 and 2.01/2.04 responses; a stateless server may be responding with 2.01 only on the first Block1 block transferred, which dominates any 2.04 responses for later blocks.

If-None-Match only works correctly on Block1 requests with (NUM=0) and MUST NOT be used on Block1 requests with NUM != 0.

3. Examples

This section gives a number of short examples with message flows for a block-wise GET, and for a PUT or POST. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and examples for the operation of the block size negotiation.

In all these examples, a Block option is shown in a decomposed way indicating the kind of Block option (1 or 2) followed by a colon, and then the block number (NUM), more bit (M), and block size exponent ($2^{*(SZX+4)}$) separated by slashes. E.g., a Block2 Option value of 33 would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

As in [RFC7252], "MID" is used as an abbreviation of "Message ID".

3.1. Block2 Examples

The first example (Figure 2) shows a GET request that is split into three blocks. The server proposes a block size of 128, and the client agrees. The first two ACKs contain a payload of 128 bytes each, and the third ACK contains a payload between 1 and 128 bytes.

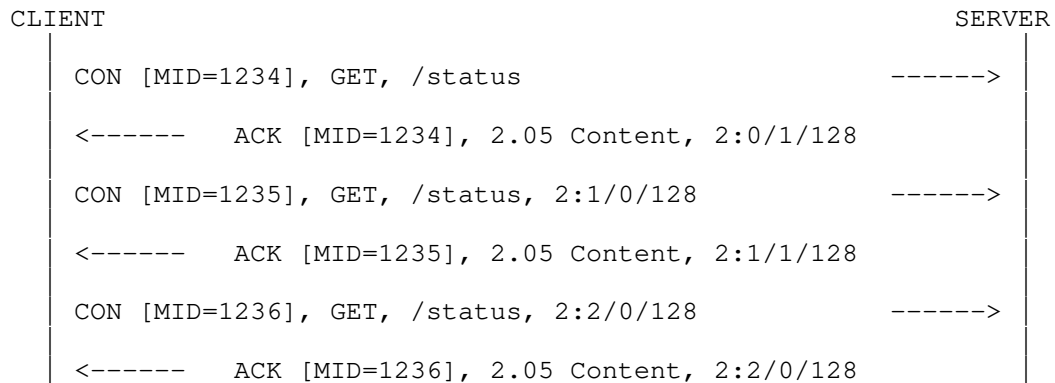


Figure 2: Simple block-wise GET

In the second example (Figure 3), the client anticipates the block-wise transfer (e.g., because of a size indication in the link-format description [RFC6690]) and sends a block size proposal. All ACK messages except for the last carry 64 bytes of payload; the last one carries between 1 and 64 bytes.

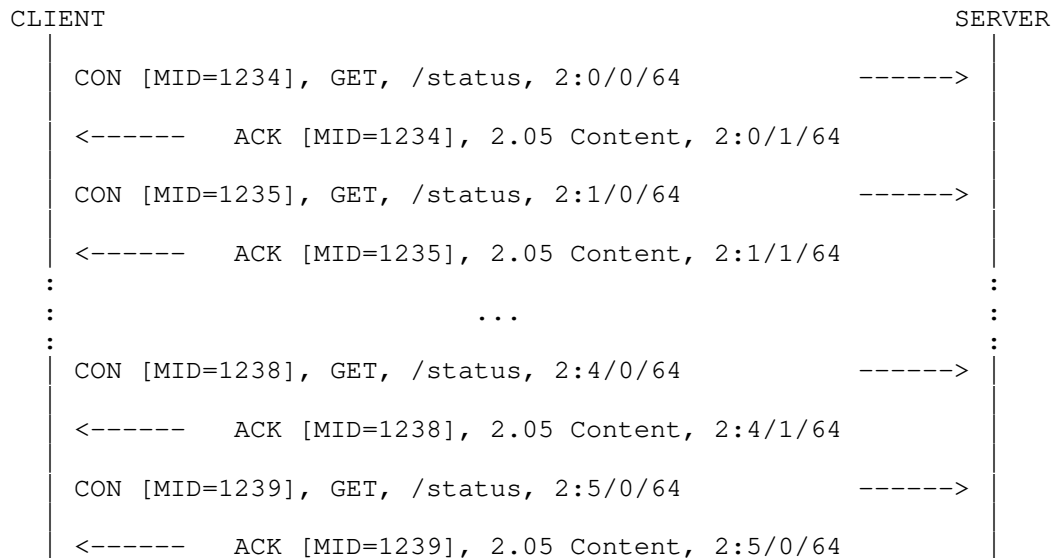


Figure 3: Block-wise GET with early negotiation

In the third example (Figure 4), the client is surprised by the need for a block-wise transfer, and unhappy with the size chosen unilaterally by the server. As it did not send a size proposal initially, the negotiation only influences the size from the second

message exchange onward. Since the client already obtained both the first and second 64-byte block in the first 128-byte exchange, it goes on requesting the third 64-byte block ("2/0/64"). None of this is (or needs to be) understood by the server, which simply responds to the requests as it best can.

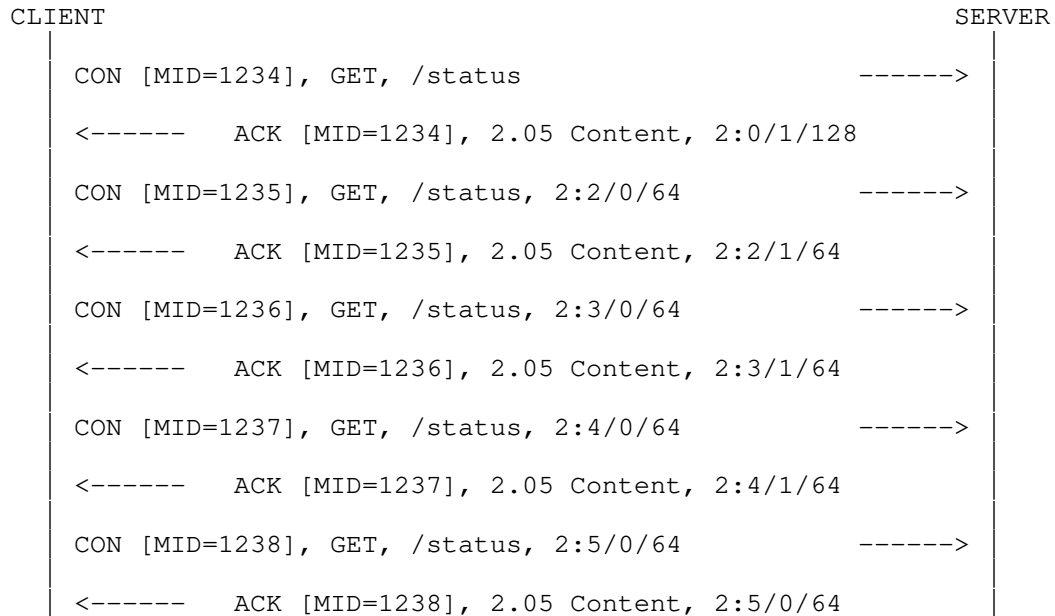


Figure 4: Block-wise GET with late negotiation

In all these (and the following) cases, retransmissions are handled by the CoAP message exchange layer, so they don't influence the block operations (Figure 5, Figure 6).

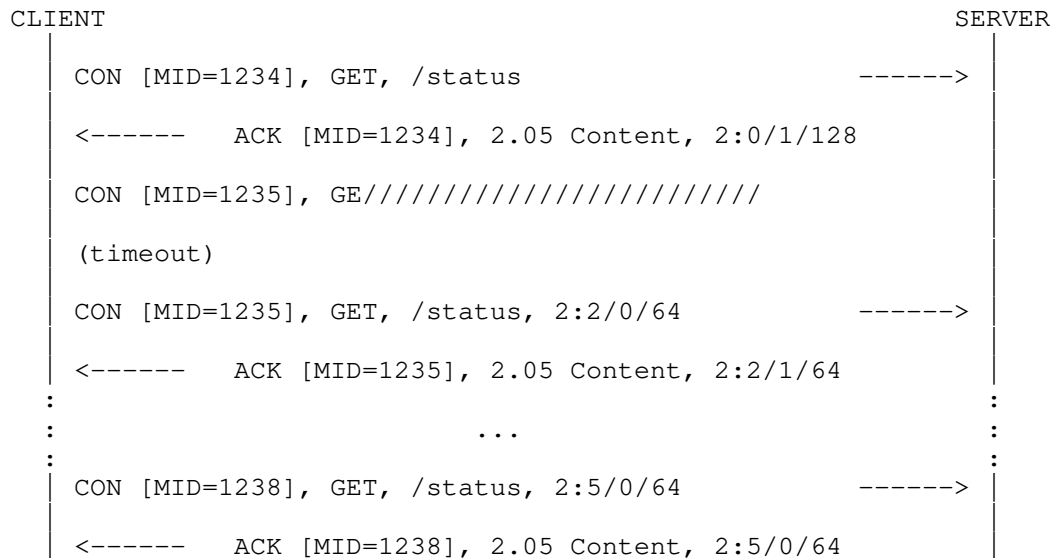


Figure 5: Block-wise GET with late negotiation and lost CON

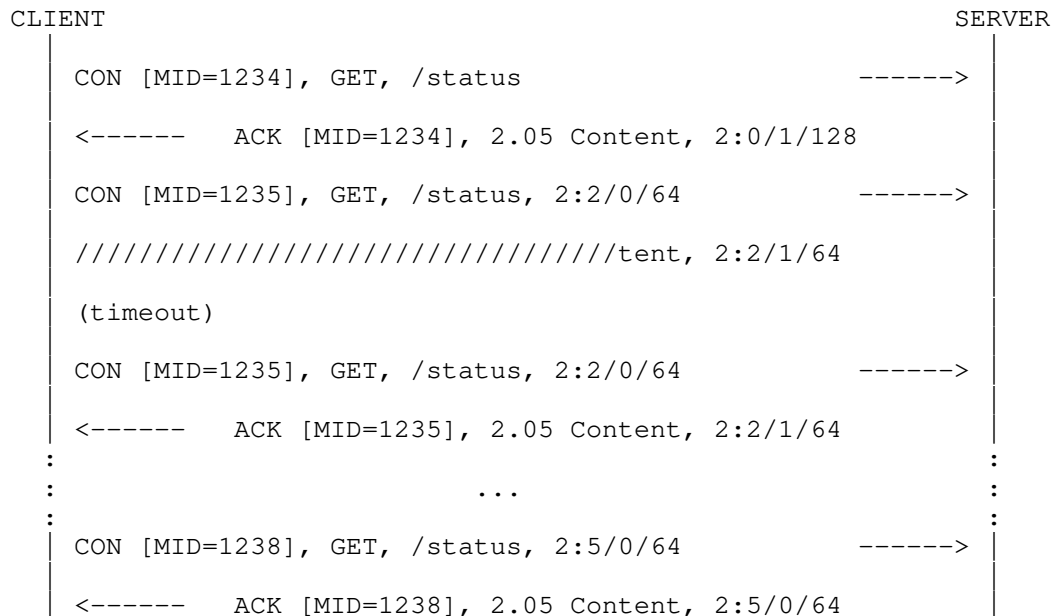


Figure 6: Block-wise GET with late negotiation and lost ACK

3.2. Block1 Examples

The following examples demonstrate a PUT exchange; a POST exchange looks the same, with different requirements on atomicity/idempotence. Note that, similar to GET, the responses to the requests that have a more bit in the request Block1 Option are provisional and carry the response code 2.31 (Continue); only the final response tells the client that the PUT did succeed.

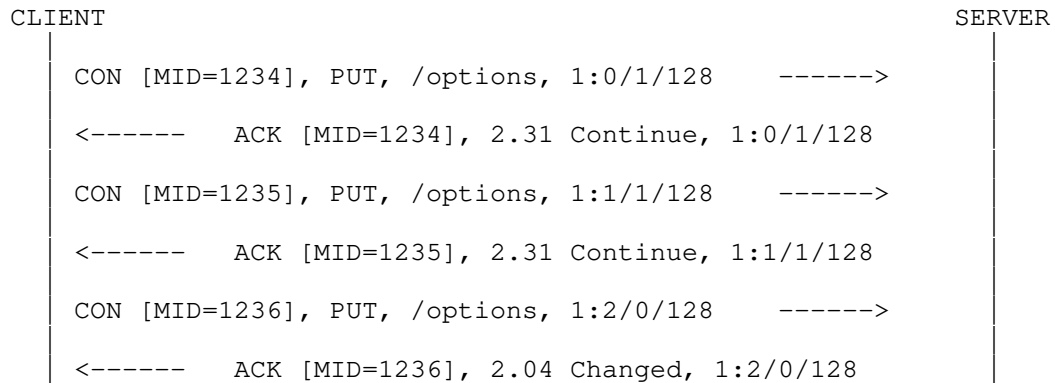


Figure 7: Simple atomic block-wise PUT

A stateless server that simply builds/updates the resource in place (statelessly) may indicate this by not setting the more bit in the response (Figure 8); in this case, the response codes are valid separately for each block being updated. This is of course only an acceptable behavior of the server if the potential inconsistency present during the run of the message exchange sequence does not lead to problems, e.g. because the resource being created or changed is not yet or not currently in use.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1:0/1/128	----->
<-----	ACK [MID=1234], 2.04 Changed, 1:0/0/128
CON [MID=1235], PUT, /options, 1:1/1/128	----->
<-----	ACK [MID=1235], 2.04 Changed, 1:1/0/128
CON [MID=1236], PUT, /options, 1:2/0/128	----->
<-----	ACK [MID=1236], 2.04 Changed, 1:2/0/128

Figure 8: Simple stateless block-wise PUT

Finally, a server receiving a block-wise PUT or POST may want to indicate a smaller block size preference (Figure 9). In this case, the client SHOULD continue with a smaller block size; if it does, it MUST adjust the block number to properly count in that smaller size.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1:0/1/128	----->
<-----	ACK [MID=1234], 2.31 Continue, 1:0/1/32
CON [MID=1235], PUT, /options, 1:4/1/32	----->
<-----	ACK [MID=1235], 2.31 Continue, 1:4/1/32
CON [MID=1236], PUT, /options, 1:5/1/32	----->
<-----	ACK [MID=1235], 2.31 Continue, 1:5/1/32
CON [MID=1237], PUT, /options, 1:6/0/32	----->
<-----	ACK [MID=1236], 2.04 Changed, 1:6/0/32

Figure 9: Simple atomic block-wise PUT with negotiation

3.3. Combining Block1 and Block2

Block options may be used in both directions of a single exchange. The following example demonstrates a block-wise POST request, resulting in a separate block-wise response.

CLIENT	SERVER
CON [MID=1234], POST, /soap, 1:0/1/128	----->
<-----	ACK [MID=1234], 2.31 Continue, 1:0/1/128
CON [MID=1235], POST, /soap, 1:1/1/128	----->
<-----	ACK [MID=1235], 2.31 Continue, 1:1/1/128
CON [MID=1236], POST, /soap, 1:2/0/128	----->
<-----	ACK [MID=1236], 2.04 Changed, 2:0/1/128, 1:2/0/128
CON [MID=1237], POST, /soap, 2:1/0/128	----->
(no payload for requests with Block2 with NUM != 0)	
(could also do late negotiation by requesting e.g. 2:2/0/64)	
<-----	ACK [MID=1237], 2.04 Changed, 2:1/1/128
CON [MID=1238], POST, /soap, 2:2/0/128	----->
<-----	ACK [MID=1238], 2.04 Changed, 2:2/1/128
CON [MID=1239], POST, /soap, 2:3/0/128	----->
<-----	ACK [MID=1239], 2.04 Changed, 2:3/0/128

Figure 10: Atomic block-wise POST with block-wise response

This model does provide for early negotiation input to the Block2 block-wise transfer, as shown below.

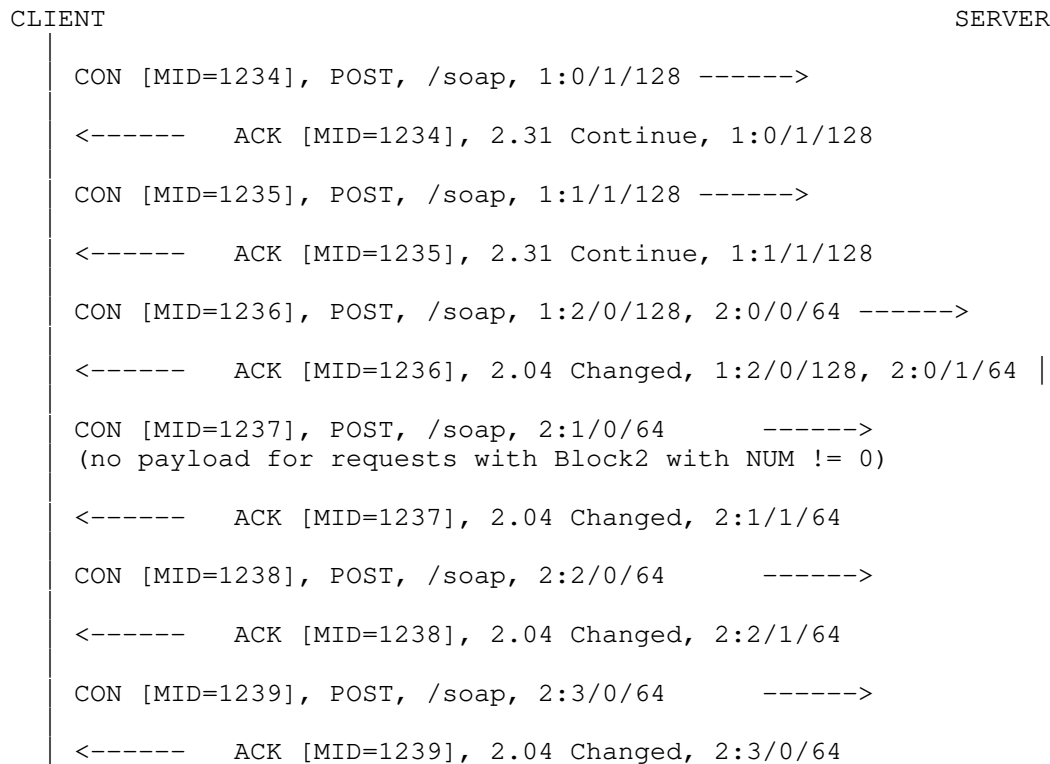
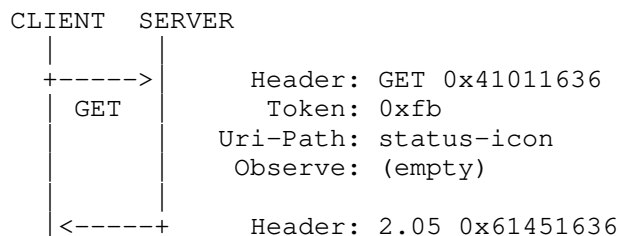


Figure 11: Atomic block-wise POST with block-wise response, early negotiation

3.4. Combining Observe and Block2

In the following example, the server first sends a direct response (Observe sequence number 62350) to the initial GET request (the resulting block-wise transfer is as in Figure 4 and has therefore been left out). The second transfer is started by a 2.05 notification that contains just the first block (Observe sequence number 62354); the client then goes on to obtain the rest of the blocks.



2.05	Token: 0xfb Block2: 0/1/128 Observe: 62350 ETag: 6f00f38e Payload: [128 bytes]
	(Usual GET transfer left out)
...	(Notification of first block:)
<-----+ 2.05	Header: 2.05 0x4145af9c Token: 0xfb Block2: 0/1/128 Observe: 62354 ETag: 6f00f392 Payload: [128 bytes]
+-- -->	Header: 0x6000af9c (Retrieval of remaining blocks)
+-----> GET	Header: GET 0x41011637 Token: 0xfc Uri-Path: status-icon Block2: 1/0/128
<-----+ 2.05	Header: 2.05 0x61451637 Token: 0xfc Block2: 1/1/128 ETag: 6f00f392 Payload: [128 bytes]
+-----> GET	Header: GET 0x41011638 Token: 0xfc Uri-Path: status-icon Block2: 2/0/128
<-----+ 2.05	Header: 2.05 0x61451638 Token: 0xfc Block2: 2/0/128 ETag: 6f00f392 Payload: [53 bytes]

Figure 12: Observe sequence with block-wise response

(Note that the choice of token 0xfc in this examples is arbitrary; tokens are just shown in this example to illustrate that the requests

for additional blocks cannot make use of the token of the Observation relationship. As a general comment on tokens, there is no other mention of tokens in this document, as block-wise transfers handle tokens like any other CoAP exchange. As usual the client is free to choose tokens for each exchange as it likes.)

In the following example, the client also uses early negotiation to limit the block size to 64 bytes.

CLIENT	SERVER
+-----> GET	Header: GET 0x41011636 Token: 0xfb Uri-Path: status-icon Observe: (empty) Block2: 0/0/64
<-----+ 2.05	Header: 2.05 0x61451636 Token: 0xfb Block2: 0/1/64 Observe: 62350 ETag: 6f00f38e Max-Age: 60 Payload: [64 bytes]
	(Usual GET transfer left out)
...	(Notification of first block:)
<-----+ 2.05	Header: 2.05 0x4145af9c Token: 0xfb Block2: 0/1/64 Observe: 62354 ETag: 6f00f392 Payload: [64 bytes]
+-- -->	Header: 0x6000af9c (Retrieval of remaining blocks)
+-----> GET	Header: GET 0x41011637 Token: 0xfc Uri-Path: status-icon Block2: 1/0/64
<-----+ 2.05	Header: 2.05 0x61451637 Token: 0xfc Block2: 1/1/64

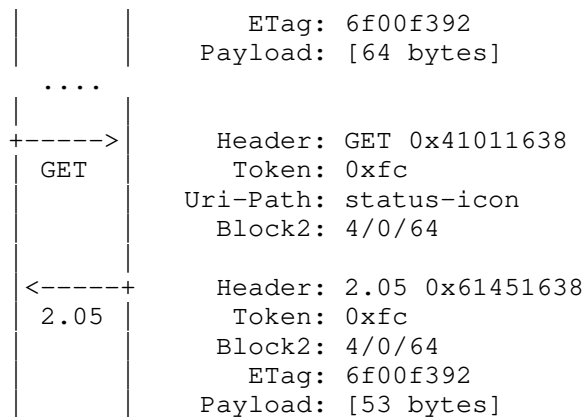


Figure 13: Observe sequence with early negotiation

4. The Size2 and Size1 Options

In many cases when transferring a large resource representation block by block, it is advantageous to know the total size early in the process. Some indication may be available from the maximum size estimate attribute "sz" provided in a resource description [RFC6690]. However, the size may vary dynamically, so a more up-to-date indication may be useful.

This specification defines two CoAP Options, Size1 for indicating the size of the representation transferred in requests, and Size2 for indicating the size of the representation transferred in responses. (Size1 has already been defined in Section 5.10.9 of [RFC7252] to provide "size information about the resource representation in a request", however that section only details the narrow case of indicating in 4.13 responses the maximum size of request payload that the server is able and willing to handle. The present specification provides details about its use as a request option as well.)

The Size2 Option may be used for two purposes:

- o in a request, to ask the server to provide a size estimate along with the usual response ("size request"). For this usage, the value MUST be set to 0.
- o in a response carrying a Block2 Option, to indicate the current estimate the server has of the total size of the resource representation, measured in bytes ("size indication").

Similarly, the Size1 Option may be used for two purposes:

- o in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation, measured in bytes ("size indication").
- o in a 4.13 response, to indicate the maximum size that would have been acceptable [RFC7252], measured in bytes.

Apart from conveying/asking for size information, the Size options have no other effect on the processing of the request or response. If the client wants to minimize the size of the payload in the resulting response, it should add a Block2 option to the request with a small block size (e.g., setting SZX=0).

The Size Options are "elective", i.e., a client MUST be prepared for the server to ignore the size estimate request. The Size Options MUST NOT occur more than once.

No.	C	U	N	R	Name	Format	Length	Default
60			x		Size1	uint	0-4	(none)
28			x		Size2	uint	0-4	(none)

Table 2: Size Option Numbers

Implementation Notes:

- o As a quality of implementation consideration, block-wise transfers for which the total size considerably exceeds the size of one block are expected to include size indications, whenever those can be provided without undue effort (preferably with the first block exchanged). If the size estimate does not change, the indication does not need to be repeated for every block.
- o The end of a block-wise transfer is governed by the M bits in the Block Options, not by exhausting the size estimates exchanged.
- o As usual for an option of type uint, the value 0 is best expressed as an empty option (0 bytes). There is no default value for either Size Option.
- o The Size Options are neither critical nor unsafe, and are marked as No-Cache-Key.

5. HTTP Mapping Considerations

In this subsection, we give some brief examples for the influence the Block options might have on intermediaries that map between CoAP and HTTP.

For mapping CoAP requests to HTTP, the intermediary may want to map the sequence of block-wise transfers into a single HTTP transfer. E.g., for a GET request, the intermediary could perform the HTTP request once the first block has been requested and could then fulfill all further block requests out of its cache. A constrained implementation may not be able to cache the entire object and may use a combination of TCP flow control and (in particular if timeouts occur) HTTP range requests to obtain the information necessary for the next block transfer at the right time.

For PUT or POST requests, historically there was more variation in how HTTP servers might implement ranges; recently, [RFC7233] has defined that Range header fields received with a request method other than GET are not to be interpreted. So, in general, the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer for these other methods within one HTTP request. If enough buffering is available, this request can be started when the last CoAP block is received. A constrained implementation may want to relieve its buffering by already starting to send the HTTP request at the time the first CoAP block is received; any HTTP 408 status code that indicates that the HTTP server became impatient with the resulting transfer can then be mapped into a CoAP 4.08 response code (similarly, 413 maps to 4.13).

For mapping HTTP to CoAP, the intermediary may want to map a single HTTP transfer into a sequence of block-wise transfers. If the HTTP client is too slow delivering a request body on a PUT or POST, the CoAP server might time out and return a 4.08 response code, which in turn maps well to an HTTP 408 status code (again, 4.13 maps to 413). HTTP range requests received on the HTTP side may be served out of a cache and/or mapped to GET requests that request a sequence of blocks overlapping the range.

(Note that, while the semantics of CoAP 4.08 and HTTP 408 differ, this difference is largely due to the different way the two protocols are mapped to transport. HTTP has an underlying TCP connection, which supplies connection state, so a HTTP 408 status code can immediately be used to indicate that a timeout occurred during transmitting a request through that active TCP connection. The CoAP 4.08 response code indicates one or more missing blocks, which may be due to timeouts or resource constraints; as there is no connection state, there is no way to deliver such a response immediately;

instead, it is delivered on the next block transfer. Still, HTTP 408 is probably the best mapping back to HTTP, as the timeout is the most likely cause for a CoAP 4.08. Note that there is no way to distinguish a timeout from a missing block for a server without creating additional state, the need for which we want to avoid.)

6. IANA Considerations

This draft adds the following option numbers to the CoAP Option Numbers registry of [RFC7252]:

Number	Name	Reference
23	Block2	[RFCXXXX]
27	Block1	[RFCXXXX]
28	Size2	[RFCXXXX]

Table 3: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [RFC7252]:

Code	Description	Reference
2.31	Continue	[RFCXXXX]
4.08	Request Entity Incomplete	[RFCXXXX]

Table 4: CoAP Response Codes

7. Security Considerations

Providing access to blocks within a resource may lead to surprising vulnerabilities. Where requests are not implemented atomically, an attacker may be able to exploit a race condition or confuse a server by inducing it to use a partially updated resource representation. Partial transfers may also make certain problematic data invisible to intrusion detection systems; it is RECOMMENDED that an intrusion detection system (IDS) that analyzes resource representations transferred by CoAP implement the Block options to gain access to entire resource representations. Still, approaches such as transferring even-numbered blocks on one path and odd-numbered blocks

on another path, or even transferring blocks multiple times with different content and obtaining a different interpretation of temporal order at the IDS than at the server, may prevent an IDS from seeing the whole picture. These kinds of attacks are well understood from IP fragmentation and TCP segmentation; CoAP does not add fundamentally new considerations.

Where access to a resource is only granted to clients making use of specific security associations, all blocks of that resource MUST be subject to the same security checks; it MUST NOT be possible for unprotected exchanges to influence blocks of an otherwise protected resource. As a related consideration, where object security is employed, PUT/POST should be implemented in the atomic fashion, unless the object security operation is performed on each access and the creation of unusable resources can be tolerated. Future end-to-end security mechanisms that may be added to CoAP itself may have related security considerations, this includes considerations about caching of blocks in clients and in proxies (see Section 2.10 and Section 5 for different strategies in performing this caching); these security considerations will need to be described in the specifications of those mechanisms.

A stateless server might be susceptible to an attack where the adversary sends a Block1 (e.g., PUT) block with a high block number: A naive implementation might exhaust its resources by creating a huge resource representation.

Misleading size indications may be used by an attacker to induce buffer overflows in poor implementations, for which the usual considerations apply.

7.1. Mitigating Resource Exhaustion Attacks

Certain block-wise requests may induce the server to create state, e.g. to create a snapshot for the block-wise GET of a fast-changing resource to enable consistent access to the same version of a resource for all blocks, or to create temporary resource representations that are collected until pressed into service by a final PUT or POST with the more bit unset. All mechanisms that induce a server to create state that cannot simply be cleaned up create opportunities for denial-of-service attacks. Servers SHOULD avoid being subject to resource exhaustion based on state created by untrusted sources. But even if this is done, the mitigation may cause a denial-of-service to a legitimate request when it is drowned out by other state-creating requests. Wherever possible, servers should therefore minimize the opportunities to create state for untrusted sources, e.g. by using stateless approaches.

Performing segmentation at the application layer is almost always better in this respect than at the transport layer or lower (IP fragmentation, adaptation layer fragmentation), for instance because there is application layer semantics that can be used for mitigation or because lower layers provide security associations that can prevent attacks. However, it is less common to apply timeouts and keepalive mechanisms at the application layer than at lower layers. Servers MAY want to clean up accumulated state by timing it out (cf. response code 4.08), and clients SHOULD be prepared to run block-wise transfers in an expedient way to minimize the likelihood of running into such a timeout.

7.2. Mitigating Amplification Attacks

[RFC7252] discusses the susceptibility of CoAP end-points for use in amplification attacks.

A CoAP server can reduce the amount of amplification it provides to an attacker by offering large resource representations only in relatively small blocks. With this, e.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

8. Acknowledgements

Much of the content of this draft is the result of discussions with the [RFC7252] authors, and via many CoRE WG discussions.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version. Esko Dijk reviewed a more recent version, leading to a number of further editorial improvements, a solution to the 4.13 ambiguity problem, and the section about combining Block and multicast. Markus Becker proposed getting rid of an ill-conceived default value for the Block2 and Block1 options. Peter Bigot insisted on a more systematic coverage of the options and response code. Qin Wu provided a review for the IETF Operational directorate, and Goeran Selander commented on the security considerations.

Kepeng Li, Linyi Tian, and Barry Leiba wrote up an early version of the Size Option, which has informed this draft. Klaus Hartke wrote some of the text describing the interaction of Block2 with Observe. Matthias Kovatsch provided a number of significant simplifications of the protocol.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

9.2. Informative References

- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <<http://www.rfc-editor.org/info/rfc4919>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<http://www.rfc-editor.org/info/rfc4944>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.

- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <<http://www.rfc-editor.org/info/rfc7233>>.

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Zach Shelby (editor)
ARM
150 Rose Orchard
San Jose, CA 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: January 9, 2017

C. Bormann
Universitaet Bremen TZI
Z. Shelby, Ed.
ARM
July 08, 2016

Block-wise transfers in CoAP
draft-ietf-core-block-21

Abstract

CoAP is a RESTful transfer protocol for constrained nodes and networks. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications will need to transfer larger payloads -- for instance, for firmware updates. With HTTP, TCP does the grunt work of slicing large payloads up into multiple packets and ensuring that they all arrive and are handled in the right order.

CoAP is based on datagram transports such as UDP or DTLS, which limits the maximum size of resource representations that can be transferred without too much fragmentation. Although UDP supports larger payloads through IP fragmentation, it is limited to 64 KiB and, more importantly, doesn't really work well for constrained applications and networks.

Instead of relying on IP fragmentation, this specification extends basic CoAP with a pair of "Block" options, for transferring multiple blocks of information from a resource representation in multiple request-response pairs. In many important cases, the Block options enable a server to be truly stateless: the server can handle each block transfer separately, with no need for a connection setup or other server-side memory of previous block transfers.

In summary, the Block options provide a minimal way to transfer larger representations in a block-wise fashion.

A CoAP implementation that does not support these options generally is limited in the size of the representations that can be exchanged. There is therefore an expectation that the Block options are very widely implemented in CoAP implementations, which is why this specification is listed as "updating" RFC 7252.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Block-wise transfers	6
2.1. The Block2 and Block1 Options	7
2.2. Structure of a Block Option	7
2.3. Block Options in Requests and Responses	9
2.4. Using the Block2 Option	11
2.5. Using the Block1 Option	13
2.6. Combining Block-wise Transfers with the Observe Option .	14
2.7. Combining Block1 and Block2	15
2.8. Combining Block2 with Multicast	15
2.9. Response Codes	16
2.9.1. 2.31 Continue	16
2.9.2. 4.08 Request Entity Incomplete	16
2.9.3. 4.13 Request Entity Too Large	16

2.10. Caching Considerations	17
3. Examples	17
3.1. Block2 Examples	18
3.2. Block1 Examples	22
3.3. Combining Block1 and Block2	23
3.4. Combining Observe and Block2	25
4. The Size2 and Size1 Options	28
5. HTTP Mapping Considerations	30
6. IANA Considerations	31
7. Security Considerations	31
7.1. Mitigating Resource Exhaustion Attacks	32
7.2. Mitigating Amplification Attacks	33
8. References	33
8.1. Normative References	33
8.2. Informative References	33
Acknowledgements	34
Authors' Addresses	35

1. Introduction

The work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes (such as microcontrollers with limited RAM and ROM [RFC7228]) and networks (such as 6LoWPAN, [RFC4944]) [RFC7252]. The CoAP protocol is intended to provide RESTful [REST] services not unlike HTTP [RFC7230], while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

This objective requires restraint in a number of sometimes conflicting ways:

- o reducing implementation complexity in order to minimize code size,
- o reducing message sizes in order to minimize the number of fragments needed for each message (in turn to maximize the probability of delivery of the message), the amount of transmission power needed and the loading of the limited-bandwidth channel,
- o reducing requirements on the environment such as stable storage, good sources of randomness or user interaction capabilities.

CoAP is based on datagram transports such as UDP, which limit the maximum size of resource representations that can be transferred without creating unreasonable levels of IP fragmentation. In addition, not all resource representations will fit into a single

link layer packet of a constrained network, which may cause adaptation layer fragmentation even if IP layer fragmentation is not required. Using fragmentation (either at the adaptation layer or at the IP layer) for the transport of larger representations would be possible up to the maximum size of the underlying datagram protocol (such as UDP), but the fragmentation/reassembly process burdens the lower layers with conversation state that is better managed in the application layer.

The present specification defines a pair of CoAP options to enable `_block-wise_` access to resource representations. The Block options provide a minimal way to transfer larger resource representations in a block-wise fashion. The overriding objective is to avoid the need for creating conversation state at the server for block-wise GET requests. (It is impossible to fully avoid creating conversation state for POST/PUT, if the creation/replacement of resources is to be atomic; where that property is not needed, there is no need to create server conversation state in this case, either.)

Block-wise transfers are realized as combinations of exchanges, each of which is performed according to the CoAP base protocol [RFC7252]. Each exchange in such a combination is governed by the specifications in [RFC7252], including the congestion control specifications (Section 4.7 of [RFC7252]) and the security considerations (Section 11 of [RFC7252]; additional security considerations then apply to the transfers as a whole, see Section 7). The present specification minimizes the constraints it adds to those base exchanges; however, not all variants of using CoAP are very useful inside a block-wise transfer (e.g., using Non-confirmable requests within block-wise transfers outside the use case of Section 2.8 would escalate the overall non-delivery probability). To be perfectly clear, the present specification also does not remove any of the constraints posed by the base specification it is strictly layered on top of; e.g., back-to-back packets are limited by Section 4.7 of [RFC7252] (NSTART as a limit for initiating exchanges, PROBING_RATE as a limit for sending with no response): block-wise transfers cannot send/solicit more traffic than a client could be sending to the same server without the block-wise mode.

In some cases, the present specification will RECOMMEND that a client perform a sequence of block-wise transfers "without undue delay". This cannot be phrased as an interoperability requirement, but is an expectation on implementation quality. Conversely, the expectation is that servers will not have go out of their way to accommodate clients that take forever to finish a block-wise transfer. E.g., for a block-wise GET, if the resource changes while this proceeds, the ETag for a further block obtained may be different. To avoid this happening all the time for a fast-changing resource, a server MAY try

to keep a cache around for a specific client for a short amount of time. The expectation here is that the lifetime for such a cache can be kept short, on the order of a few expected round-trip times, counting from the previous block transferred.

In summary, this specification adds a pair of Block options to CoAP that can be used for block-wise transfers. Benefits of using these options include:

- o Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- o No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- o The transfer of each block is acknowledged, enabling individual retransmission if required.
- o Both sides have a say in the block size that actually will be used.
- o The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- o If needed, the Block options can also be used (without changes) to provide random access to power-of-two sized blocks within a resource representation.

A CoAP implementation that does not support these options generally is limited in the size of the representations that can be exchanged, see Section 4.6 of [RFC7252]. Even though the options are Critical, a server may decide to start using them in an unsolicited way in a response. No effort was expended to provide a capability indication mechanism supporting that decision: since the block-wise transfer mechanisms are so fundamental to the use of CoAP for representations larger than about a kilobyte, there is an expectation that they are very widely implemented.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119] and indicate requirement levels for compliant CoAP implementations.

In this document, the term "byte" is used in its now customary sense as a synonym for "octet".

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C, except that the operator "***" stands for exponentiation.

2. Block-wise transfers

As discussed in the introduction, there are good reasons to limit the size of datagrams in constrained networks:

- o by the maximum datagram size (~ 64 KiB for UDP)
- o by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- o by the desire to avoid adaptation layer fragmentation (60-80 bytes for 6LoWPAN [RFC4919])

When a resource representation is larger than can be comfortably transferred in the payload of a single CoAP datagram, a Block option can be used to indicate a block-wise transfer. As payloads can be sent both with requests and with responses, this specification provides two separate options for each direction of payload transfer. In naming these options (for block-wise transfers as well as in Section 4), we use the number 1 ("Block1", "Size1") to refer to the transfer of the resource representation that pertains to the request, and the number 2 ("Block2", "Size2") to refer to the transfer of the resource representation for the response.

In the following, the term "payload" will be used for the actual content of a single CoAP message, i.e. a single block being transferred, while the term "body" will be used for the entire resource representation that is being transferred in a block-wise fashion. The Content-Format option applies to the body, not to the payload, in particular the boundaries between the blocks may be in places that are not separating whole units in terms of the structure, encoding, or content-coding used by the Content-Format. (Similarly, the ETag option defined in Section 5.10.6 of [RFC7252] applies to the whole representation of the resource and thus to the body of the response.)

In most cases, all blocks being transferred for a body (except for the last one) will be of the same size. (If the first request uses a bigger block size than the receiver prefers, subsequent requests will use the preferred block size.) The block size is not fixed by the protocol. To keep the implementation as simple as possible, the Block options support only a small range of power-of-two block sizes, from 2^4 (16) to 2^{10} (1024) bytes. As bodies often will not evenly divide into the power-of-two block size chosen, the size need not be reached in the final block (but even for the final block, the

chosen power-of-two size will still be indicated in the block size field of the Block option).

2.1. The Block2 and Block1 Options

No.	C	U	N	R	Name	Format	Length	Default
23	C	U	-	-	Block2	uint	0-3	(none)
27	C	U	-	-	Block1	uint	0-3	(none)

Table 1: Block Option Numbers

Both Block1 and Block2 options can be present both in request and response messages. In either case, the Block1 Option pertains to the request payload, and the Block2 Option pertains to the response payload.

Hence, for the methods defined in [RFC7252], Block1 is useful with the payload-bearing POST and PUT requests and their responses. Block2 is useful with GET, POST, and PUT requests and their payload-bearing responses (2.01, 2.02, 2.04, 2.05 -- see Section 5.5 of [RFC7252]).

Where Block1 is present in a request or Block2 in a response (i.e., in that message to the payload of which it pertains) it indicates a block-wise transfer and describes how this specific block-wise payload forms part of the entire body being transferred ("descriptive usage"). Where it is present in the opposite direction, it provides additional control on how that payload will be formed or was processed ("control usage").

Implementation of either Block option is intended to be optional. However, when it is present in a CoAP message, it MUST be processed (or the message rejected); therefore it is identified as a critical option. It MUST NOT occur more than once.

2.2. Structure of a Block Option

Three items of information may need to be transferred in a Block (Block1 or Block2) option:

- o The size of the block (SZX);
- o whether more blocks are following (M);

- o the relative number of the block (NUM) within a sequence of blocks with the given size.

The value of the Block Option is a variable-size (0 to 3 byte) unsigned integer (uint, see Section 3.2 of [RFC7252]). This integer value encodes these three fields, see Figure 1. (Due to the CoAP uint encoding rules, when all of NUM, M, and SZX happen to be zero, a zero-byte integer will be sent.)

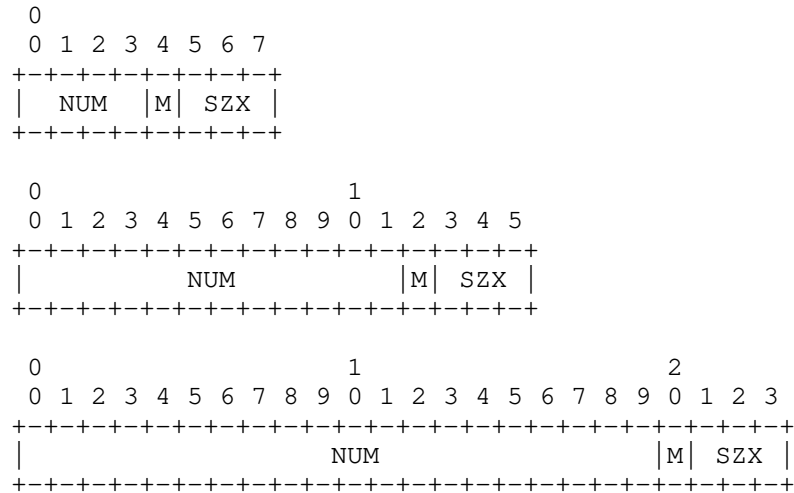


Figure 1: Block option value

The block size is encoded using a three-bit unsigned integer (0 for $2^{**}4$ to 6 for $2^{**}10$ bytes), which we call the "SZX" ("size exponent"); the actual block size is then $2^{**}(\text{SZX} + 4)$. SZX is transferred in the three least significant bits of the option value (i.e., $\text{val} \& 7$ where "val" is the value of the option).

The fourth least significant bit, the M or "more" bit ($\text{val} \& 8$), indicates whether more blocks are following or the current block-wise transfer is the last block being transferred.

The option value divided by sixteen (the NUM field) is the sequence number of the block currently being transferred, starting from zero. The current transfer is therefore about the "size" bytes starting at byte $\text{NUM} \ll (\text{SZX} + 4)$.

Implementation note: As an implementation convenience, $(\text{val} \& \sim 0xF) \ll (\text{val} \& 7)$, i.e., the option value with the last 4 bits masked out, shifted to the left by the value of SZX, gives the byte position of the first byte of the block being transferred.

More specifically, within the option value of a Block1 or Block2 Option, the meaning of the option fields is defined as follows:

NUM: Block Number, indicating the block number being requested or provided. Block number 0 indicates the first block of a body (i.e., starting with the first byte of the body).

M: More Flag ("not last block"). For descriptive usage, this flag, if unset, indicates that the payload in this message is the last block in the body; when set it indicates that there are one or more additional blocks available. When a Block2 Option is used in a request to retrieve a specific block number ("control usage"), the M bit MUST be sent as zero and ignored on reception. (In a Block1 Option in a response, the M flag is used to indicate atomicity, see below.)

SZX: Block Size. The block size is represented as three-bit unsigned integer indicating the size of a block to the power of two. Thus block size = $2^{(SZX + 4)}$. The allowed values of SZX are 0 to 6, i.e., the minimum block size is $2^{(0+4)} = 16$ and the maximum is $2^{(6+4)} = 1024$. The value 7 for SZX (which would indicate a block size of 2048) is reserved, i.e. MUST NOT be sent and MUST lead to a 4.00 Bad Request response code upon reception in a request.

There is no default value for the Block1 and Block2 Options. Absence of one of these options is equivalent to an option value of 0 with respect to the value of NUM and M that could be given in the option, i.e. it indicates that the current block is the first and only block of the transfer (block number 0, M bit not set). However, in contrast to the explicit value 0, which would indicate an SZX of 0 and thus a size value of 16 bytes, there is no specific explicit size implied by the absence of the option -- the size is left unspecified. (As for any uint, the explicit value 0 is efficiently indicated by a zero-length option; this, therefore, is different in semantics from the absence of the option.)

2.3. Block Options in Requests and Responses

The Block options are used in one of three roles:

- o In descriptive usage, i.e., a Block2 Option in a response (such as a 2.05 response for GET), or a Block1 Option in a request (a PUT or POST):
 - * The NUM field in the option value describes what block number is contained in the payload of this message.

- * The M bit indicates whether further blocks need to be transferred to complete the transfer of that body.
 - * The block size implied by SZX MUST match the size of the payload in bytes, if the M bit is set. (SZX does not govern the payload size if M is unset). For Block2, if the request suggested a larger value of SZX, the next request MUST move SZX down to the size given in the response. (The effect is that, if the server uses the smaller of (1) its preferred block size and (2) the block size requested, all blocks for a body use the same block size.)
- o A Block2 Option in control usage in a request (e.g., GET):
 - * The NUM field in the Block2 Option gives the block number of the payload that is being requested to be returned in the response.
 - * In this case, the M bit has no function and MUST be set to zero.
 - * The block size given (SZX) suggests a block size (in the case of block number 0) or repeats the block size of previous blocks received (in the case of a non-zero block number).
 - o A Block1 Option in control usage in a response (e.g., a 2.xx response for a PUT or POST request):
 - * The NUM field of the Block1 Option indicates what block number is being acknowledged.
 - * If the M bit was set in the request, the server can choose whether to act on each block separately, with no memory, or whether to handle the request for the entire body atomically, or any mix of the two.
 - + If the M bit is also set in the response, it indicates that this response does not carry the final response code to the request, i.e. the server collects further blocks from the same endpoint and plans to implement the request atomically (e.g., acts only upon reception of the last block of payload). In this case, the response MUST NOT carry a Block2 option.
 - + Conversely, if the M bit is unset even though it was set in the request, it indicates the block-wise request was enacted now specifically for this block, and the response carries the final response to this request (and to any previous ones)

with the M bit set in the response's Block1 Option in this sequence of block-wise transfers); the client is still expected to continue sending further blocks, the request method for which may or may not also be enacted per-block. (Note that the resource is now in a partially updated state; this approach is only appropriate where exposing such an intermediate state is acceptable. The client can reduce the window by quickly continuing to update the resource, or, in case of failure, restarting the update.)

- * Finally, the SZX block size given in a control Block1 Option indicates the largest block size preferred by the server for transfers toward the resource that is the same or smaller than the one used in the initial exchange; the client SHOULD use this block size or a smaller one in all further requests in the transfer sequence, even if that means changing the block size (and possibly scaling the block number accordingly) from now on.

Using one or both Block options, a single REST operation can be split into multiple CoAP message exchanges. As specified in [RFC7252], each of these message exchanges uses their own CoAP Message ID.

The Content-Format Option sent with the requests or responses MUST reflect the content-format of the entire body. If blocks of a response body arrive with different content-format options, it is up to the client how to handle this error (it will typically abort any ongoing block-wise transfer). If blocks of a request arrive at a server with mismatching content-format options, the server MUST NOT assemble them into a single request; this usually leads to a 4.08 (Request Entity Incomplete, Section 2.9.2) error response on the mismatching block.

2.4. Using the Block2 Option

When a request is answered with a response carrying a Block2 Option with the M bit set, the requester may retrieve additional blocks of the resource representation by sending further requests with the same options as the initial request and a Block2 Option giving the block number and block size desired. In a request, the client MUST set the M bit of a Block2 Option to zero and the server MUST ignore it on reception.

To influence the block size used in a response, the requester MAY also use the Block2 Option on the initial request, giving the desired size, a block number of zero and an M bit of zero. A server MUST use the block size indicated or a smaller size. Any further block-wise requests for blocks beyond the first one MUST indicate the same block

size that was used by the server in the response for the first request that gave a desired size using a Block2 Option.

Once the Block2 Option is used by the requester and a first response has been received with a possibly adjusted block size, all further requests in a single block-wise transfer will ultimately converge on using the same size, except that there may not be enough content to fill the last block (the one returned with the M bit not set). (Note that the client may start using the Block2 Option in a second request after a first request without a Block2 Option resulted in a Block2 option in the response.) The server uses the block size indicated in the request option or a smaller size, but the requester **MUST** take note of the actual block size used in the response it receives to its initial request and proceed to use it in subsequent requests. The server behavior **MUST** ensure that this client behavior results in the same block size for all responses in a sequence (except for the last one with the M bit not set, and possibly the first one if the initial request did not contain a Block2 Option).

Block-wise transfers can be used to GET resources the representations of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing resources. In the latter case, the Block2 Option **SHOULD** be used in conjunction with the ETag Option ([RFC7252], Section 5.10.6), to ensure that the blocks being reassembled are from the same version of the representation: The server **SHOULD** include an ETag option in each response. If an ETag option is available, the client, when reassembling the representation from the blocks being exchanged, **MUST** compare ETag Options. If the ETag Options do not match in a GET transfer, the requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server **MAY** cache the current value of a representation for an ongoing sequence of requests. (The server may identify the sequence by the combination of the requesting end-point and the URI being the same in each block-wise request.) Note well that this specification makes no requirement for the server to establish any state; however, servers that offer quickly changing resources may thereby make it impossible for a client to ever retrieve a consistent set of blocks. Clients that want to retrieve all blocks of a resource **SHOULD** strive to do so without undue delay. Servers can fully expect to be free to discard any cached state after a period of `EXCHANGE_LIFETIME` ([RFC7252], Section 4.8.2) after the last access to the state, however, there is no requirement to always keep the state for as long.

The Block2 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise response payload transfer (e.g., GET) operations to the same resource. This is rarely a

requirement, but as a workaround, a client may vary the cache key (e.g., by using one of several URIs accessing resources with the same semantics, or by varying a proxy-safe elective option).

2.5. Using the Block1 Option

In a request with a request payload (e.g., PUT or POST), the Block1 Option refers to the payload in the request (descriptive usage).

In response to a request with a payload (e.g., a PUT or POST transfer), the block size given in the Block1 Option indicates the block size preference of the server for this resource (control usage). Obviously, at this point the first block has already been transferred by the client without benefit of this knowledge. Still, the client SHOULD heed the preference indicated and, for all further blocks, use the block size preferred by the server or a smaller one. Note that any reduction in the block size may mean that the second request starts with a block number larger than one, as the first request already transferred multiple blocks as counted in the smaller size.

To counter the effects of adaptation layer fragmentation on packet delivery probability, a client may want to give up retransmitting a request with a relatively large payload even before MAX_RETRANSMIT has been reached, and try restating the request as a block-wise transfer with a smaller payload. Note that this new attempt is then a new message-layer transaction and requires a new Message ID. (Because of the uncertainty whether the request or the acknowledgement was lost, this strategy is useful mostly for idempotent requests.)

In a block-wise transfer of a request payload (e.g., a PUT or POST) that is intended to be implemented in an atomic fashion at the server, the actual creation/replacement takes place at the time the final block, i.e. a block with the M bit unset in the Block1 Option, is received. In this case, all success responses to non-final blocks carry the response code 2.31 (Continue, Section 2.9.1). If not all previous blocks are available at the server at the time of processing the final block, the transfer fails and error code 4.08 (Request Entity Incomplete, Section 2.9.2) MUST be returned. A server MAY also return a 4.08 error code for any (final or non-final) Block1 transfer that is not in sequence; clients that do not have specific mechanisms to handle this case therefore SHOULD always start with block zero and send the following blocks in order.

One reason that a client might encounter a 4.08 error code is that the server has already timed out and discarded the partial request body being assembled. Clients SHOULD strive to send all blocks of a

request without undue delay. Servers can fully expect to be free to discard any partial request body when a period of `EXCHANGE_LIFETIME` ([RFC7252], Section 4.8.2) has elapsed after the most recent block was transferred; however, there is no requirement on a server to always keep the partial request body for as long.

The error code 4.13 (Request Entity Too Large) can be returned at any time by a server that does not currently have the resources to store blocks for a block-wise request payload transfer that it would intend to implement in an atomic fashion. (Note that a 4.13 response to a request that does not employ Block1 is a hint for the client to try sending Block1, and a 4.13 response with a smaller SZX in its Block1 option than requested is a hint to try a smaller SZX.)

A block-wise transfer of a request payload that is implemented in a stateless fashion at the server is likely to leave the resource being operated on in an inconsistent state during the time the transfer is still ongoing or when the client does not complete the transfer. This characteristic is closer to that of remote file systems than to that of HTTP, where state is always kept on the server during a transfer. Techniques well known from shared file access (e.g., client-specific temporary resources) can be used to mitigate this difference from HTTP.

The Block1 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations to the same resource. Starting a new block-wise sequence of requests to the same resource (before an old sequence from the same endpoint was finished) simply overwrites the context the server may still be keeping. (This is probably exactly what one wants in this case -- the client may simply have restarted and lost its knowledge of the previous sequence.)

2.6. Combining Block-wise Transfers with the Observe Option

The Observe Option provides a way for a client to be notified about changes over time of a resource [RFC7641]. Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. The following rules apply to the combination of block-wise transfers with notifications.

Observation relationships always apply to an entire resource; the Block2 option does not provide a way to observe a single block of a resource.

As with basic GET transfers, the client can indicate its desired block size in a Block2 Option in the GET request establishing or renewing the observation relationship. If the server supports block-

wise transfers, it SHOULD take note of the block size and apply it as a maximum size to all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the entry in that list is updated by the server receiving a new GET request for the resource from the client).

When sending a 2.05 (Content) notification, the server only sends the first block of the representation. The client retrieves the rest of the representation as if it had caused this first response by a GET request, i.e., by using additional GET requests with Block2 options containing NUM values greater than zero. (This results in the transfer of the entire representation, even if only some of the blocks have changed with respect to a previous notification.)

As with other dynamically changing resources, to ensure that the blocks being reassembled are from the same version of the representation, the server SHOULD include an ETag option in each response, and the reassembling client MUST compare the ETag options (Section 2.4). Even more so than for the general case of Block2, clients that want to retrieve all blocks of a resource they have been notified about with a first block SHOULD strive to do so without undue delay.

See Section 3.4 for examples.

2.7. Combining Block1 and Block2

In PUT and particularly in POST exchanges, both the request body and the response body may be large enough to require the use of block-wise transfers. First, the Block1 transfer of the request body proceeds as usual. In the exchange of the last slice of this block-wise transfer, the response carries the first slice of the Block2 transfer (NUM is zero). To continue this Block2 transfer, the client continues to send requests similar to the requests in the Block1 phase, but leaves out the Block1 options and includes a Block2 request option with non-zero NUM.

Block2 transfers that retrieve the response body for a request that used Block1 MUST be performed in sequential order.

2.8. Combining Block2 with Multicast

A client can use the Block2 option in a multicast GET request with NUM = 0 to aid in limiting the size of the response.

Similarly, a response to a multicast GET request can use a Block2 option with NUM = 0 if the representation is large, or to further limit the size of the response.

In both cases, the client retrieves any further blocks using unicast exchanges; in the unicast requests, the client SHOULD heed any block size preferences indicated by the server in the response to the multicast request.

Other uses of the Block options in conjunction with multicast messages are for further study.

2.9. Response Codes

Two response codes are defined by this specification beyond those already defined in [RFC7252], and another response code is extended in its meaning.

2.9.1. 2.31 Continue

This new success status code indicates that the transfer of this block of the request body was successful and that the server encourages sending further blocks, but that a final outcome of the whole block-wise request cannot yet be determined. No payload is returned with this response code.

2.9.2. 4.08 Request Entity Incomplete

This new client error status code indicates that the server has not received the blocks of the request body that it needs to proceed. The client has not sent all blocks, not sent them in the order required by the server, or has sent them long enough ago that the server has already discarded them.

(Note that one reason for not having the necessary blocks at hand may be a Content-Format mismatch, see Section 2.3. Implementation note: A server can reject a Block1 transfer with this code when NUM != 0 and a different Content-Format is indicated than expected from the current state of the resource. If it implements the transfer in a stateless fashion, it can match up the Content-Format of the block against that of the existing resource. If it implements the transfer in an atomic fashion, it can match up the block against the partially reassembled piece of representation that is going to replace the state of the resource.)

2.9.3. 4.13 Request Entity Too Large

In [RFC7252], Section 5.9.2.9, the response code 4.13 (Request Entity Too Large) is defined to be like HTTP 413 "Request Entity Too Large". [RFC7252] also recommends that this response SHOULD include a Size1 Option (Section 4) to indicate the maximum size of request entity the

server is able and willing to handle, unless the server is not in a position to make this information available.

The present specification allows the server to return this response code at any time during a Block1 transfer to indicate that it does not currently have the resources to store blocks for a transfer that it would intend to implement in an atomic fashion. It also allows the server to return a 4.13 response to a request that does not employ Block1 as a hint for the client to try sending Block1. Finally, a 4.13 response to a request with a Block1 option (control usage, see Section 2.3) where the response carries a smaller SZX in its Block1 option is a hint to try that smaller SZX.

2.10. Caching Considerations

This specification attempts to leave a variety of implementation strategies open for caches, in particular those in caching proxies. E.g., a cache is free to cache blocks individually, but also could wait to obtain the complete representation before it serves parts of it. Partial caching may be more efficient in a cross-proxy (equivalent to a streaming HTTP proxy). A cached block (partial cached response) can be used in place of a complete response to satisfy a block-wise request that is presented to a cache. Note that different blocks can have different Max-Age values, as they are transferred at different times. A response with a block updates the freshness of the complete representation. Individual blocks can be validated, and validating a single block validates the complete representation. A response with a Block1 Option in control usage with the M bit set invalidates cached responses for the target URI.

A cache or proxy that combines responses (e.g., to split blocks in a request or increase the block size in a response, or a cross-proxy) may need to combine 2.31 and 2.01/2.04 responses; a stateless server may be responding with 2.01 only on the first Block1 block transferred, which dominates any 2.04 responses for later blocks.

If-None-Match only works correctly on Block1 requests with (NUM=0) and MUST NOT be used on Block1 requests with NUM != 0.

3. Examples

This section gives a number of short examples with message flows for a block-wise GET, and for a PUT or POST. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and examples for the operation of the block size negotiation.

In all these examples, a Block option is shown in a decomposed way indicating the kind of Block option (1 or 2) followed by a colon, and then the block number (NUM), more bit (M), and block size exponent ($2^{*(SZX+4)}$) separated by slashes. E.g., a Block2 Option value of 33 would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

As in [RFC7252], "MID" is used as an abbreviation of "Message ID".

3.1. Block2 Examples

The first example (Figure 2) shows a GET request that is split into three blocks. The server proposes a block size of 128, and the client agrees. The first two ACKs contain a payload of 128 bytes each, and the third ACK contains a payload between 1 and 128 bytes.

CLIENT	SERVER
CON [MID=1234], GET, /status	----->
<----- ACK [MID=1234], 2.05 Content, 2:0/1/128	
CON [MID=1235], GET, /status, 2:1/0/128	----->
<----- ACK [MID=1235], 2.05 Content, 2:1/1/128	
CON [MID=1236], GET, /status, 2:2/0/128	----->
<----- ACK [MID=1236], 2.05 Content, 2:2/0/128	

Figure 2: Simple block-wise GET

In the second example (Figure 3), the client anticipates the block-wise transfer (e.g., because of a size indication in the link-format description [RFC6690]) and sends a block size proposal. All ACK messages except for the last carry 64 bytes of payload; the last one carries between 1 and 64 bytes.

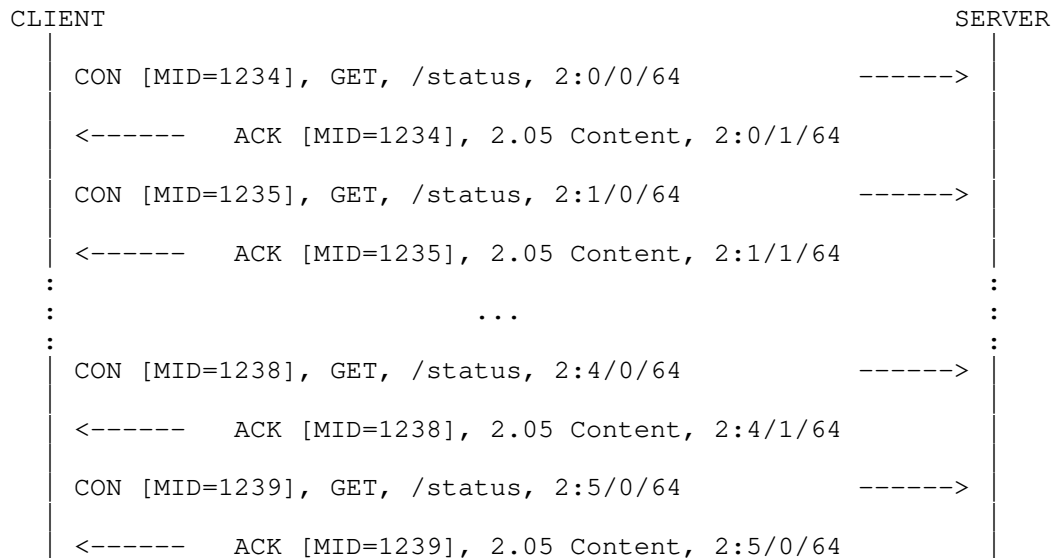


Figure 3: Block-wise GET with early negotiation

In the third example (Figure 4), the client is surprised by the need for a block-wise transfer, and unhappy with the size chosen unilaterally by the server. As it did not send a size proposal initially, the negotiation only influences the size from the second message exchange onward. Since the client already obtained both the first and second 64-byte block in the first 128-byte exchange, it goes on requesting the third 64-byte block ("2/0/64"). None of this is (or needs to be) understood by the server, which simply responds to the requests as it best can.

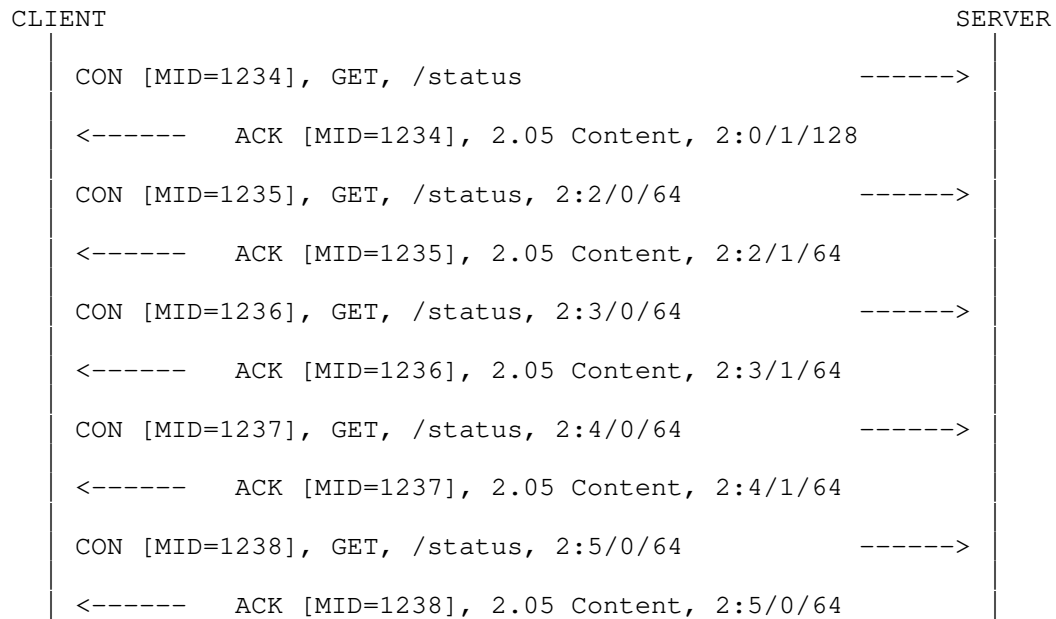


Figure 4: Block-wise GET with late negotiation

In all these (and the following) cases, retransmissions are handled by the CoAP message exchange layer, so they don't influence the block operations (Figure 5, Figure 6).

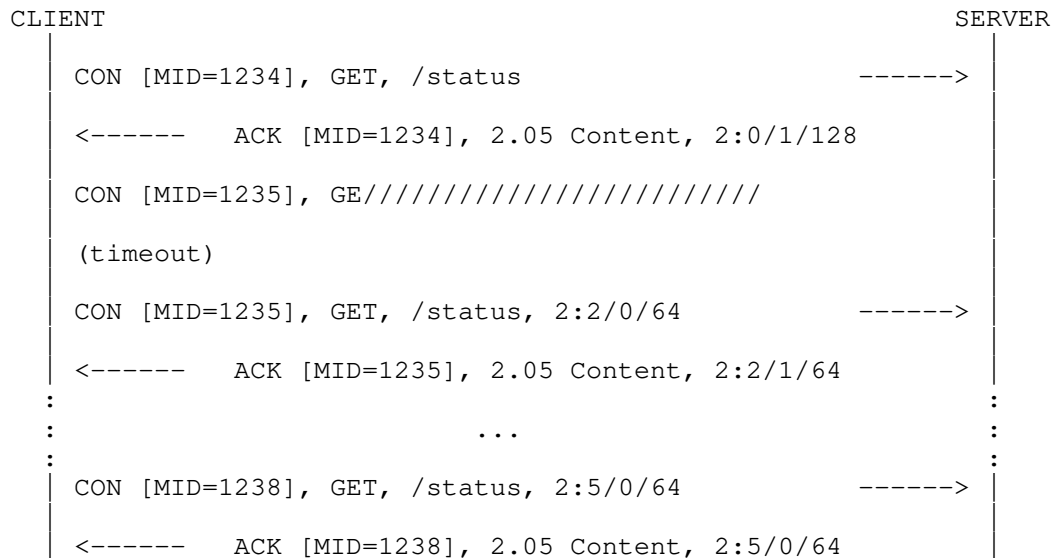


Figure 5: Block-wise GET with late negotiation and lost CON

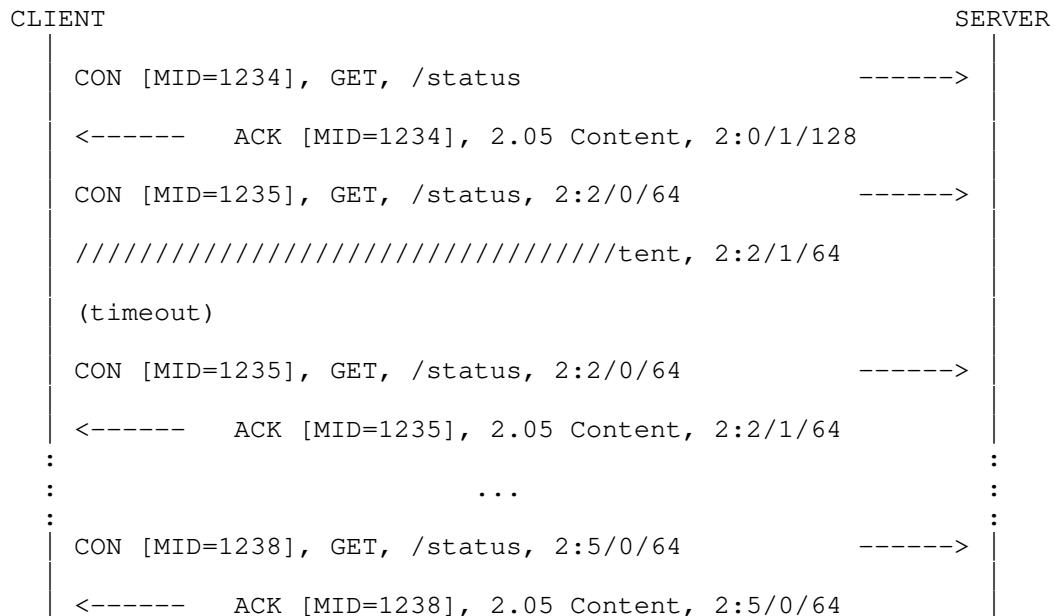


Figure 6: Block-wise GET with late negotiation and lost ACK

3.2. Block1 Examples

The following examples demonstrate a PUT exchange; a POST exchange looks the same, with different requirements on atomicity/idempotence. Note that, similar to GET, the responses to the requests that have a more bit in the request Block1 Option are provisional and carry the response code 2.31 (Continue); only the final response tells the client that the PUT did succeed.

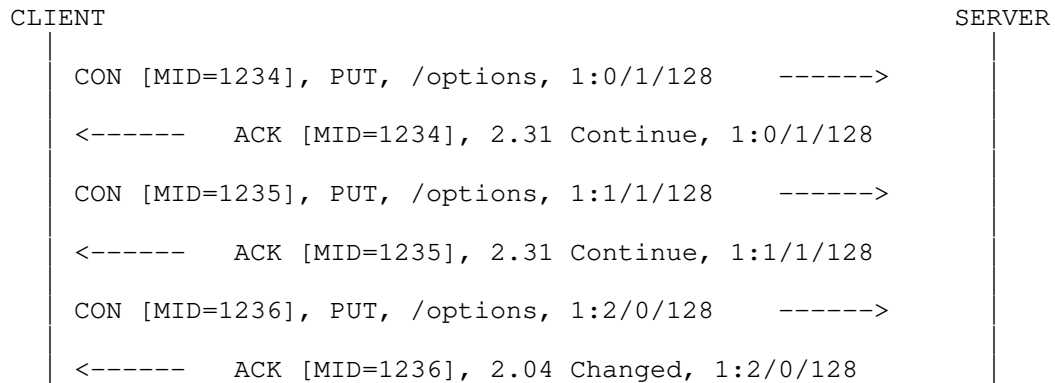


Figure 7: Simple atomic block-wise PUT

A stateless server that simply builds/updates the resource in place (statelessly) may indicate this by not setting the more bit in the response (Figure 8); in this case, the response codes are valid separately for each block being updated. This is of course only an acceptable behavior of the server if the potential inconsistency present during the run of the message exchange sequence does not lead to problems, e.g. because the resource being created or changed is not yet or not currently in use.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1:0/1/128	----->
<-----	ACK [MID=1234], 2.04 Changed, 1:0/0/128
CON [MID=1235], PUT, /options, 1:1/1/128	----->
<-----	ACK [MID=1235], 2.04 Changed, 1:1/0/128
CON [MID=1236], PUT, /options, 1:2/0/128	----->
<-----	ACK [MID=1236], 2.04 Changed, 1:2/0/128

Figure 8: Simple stateless block-wise PUT

Finally, a server receiving a block-wise PUT or POST may want to indicate a smaller block size preference (Figure 9). In this case, the client SHOULD continue with a smaller block size; if it does, it MUST adjust the block number to properly count in that smaller size.

CLIENT	SERVER
CON [MID=1234], PUT, /options, 1:0/1/128	----->
<-----	ACK [MID=1234], 2.31 Continue, 1:0/1/32
CON [MID=1235], PUT, /options, 1:4/1/32	----->
<-----	ACK [MID=1235], 2.31 Continue, 1:4/1/32
CON [MID=1236], PUT, /options, 1:5/1/32	----->
<-----	ACK [MID=1235], 2.31 Continue, 1:5/1/32
CON [MID=1237], PUT, /options, 1:6/0/32	----->
<-----	ACK [MID=1236], 2.04 Changed, 1:6/0/32

Figure 9: Simple atomic block-wise PUT with negotiation

3.3. Combining Block1 and Block2

Block options may be used in both directions of a single exchange. The following example demonstrates a block-wise POST request, resulting in a separate block-wise response.

CLIENT	SERVER
CON [MID=1234], POST, /soap, 1:0/1/128	----->
<----- ACK [MID=1234], 2.31 Continue, 1:0/1/128	
CON [MID=1235], POST, /soap, 1:1/1/128	----->
<----- ACK [MID=1235], 2.31 Continue, 1:1/1/128	
CON [MID=1236], POST, /soap, 1:2/0/128	----->
<----- ACK [MID=1236], 2.04 Changed, 2:0/1/128, 1:2/0/128	
CON [MID=1237], POST, /soap, 2:1/0/128	----->
(no payload for requests with Block2 with NUM != 0)	
(could also do late negotiation by requesting e.g. 2:2/0/64)	
<----- ACK [MID=1237], 2.04 Changed, 2:1/1/128	
CON [MID=1238], POST, /soap, 2:2/0/128	----->
<----- ACK [MID=1238], 2.04 Changed, 2:2/1/128	
CON [MID=1239], POST, /soap, 2:3/0/128	----->
<----- ACK [MID=1239], 2.04 Changed, 2:3/0/128	

Figure 10: Atomic block-wise POST with block-wise response

This model does provide for early negotiation input to the Block2 block-wise transfer, as shown below.

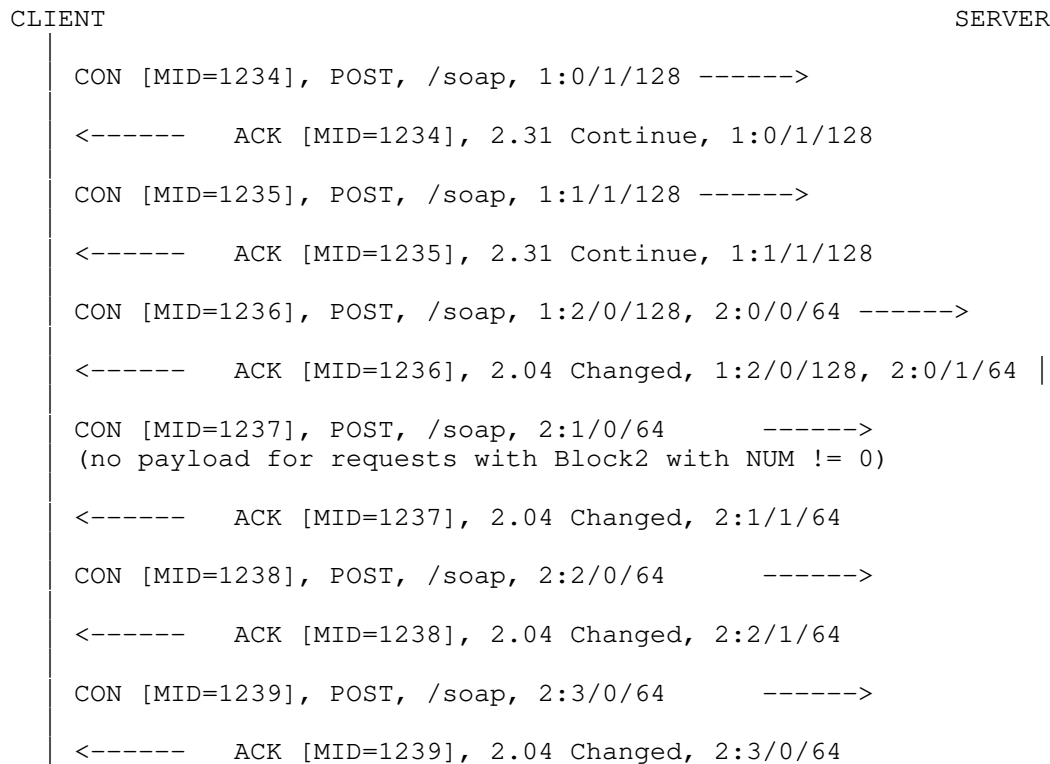
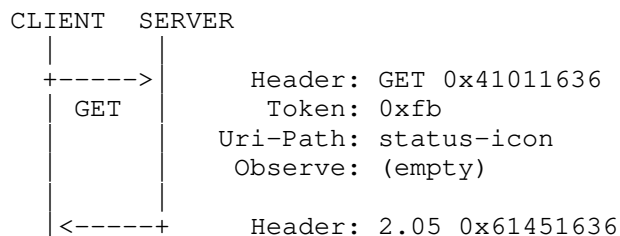


Figure 11: Atomic block-wise POST with block-wise response, early negotiation

3.4. Combining Observe and Block2

In the following example, the server first sends a direct response (Observe sequence number 62350) to the initial GET request (the resulting block-wise transfer is as in Figure 4 and has therefore been left out). The second transfer is started by a 2.05 notification that contains just the first block (Observe sequence number 62354); the client then goes on to obtain the rest of the blocks.



```

| 2.05 | Token: 0xfb
|      | Block2: 0/1/128
|      | Observe: 62350
|      | ETag: 6f00f38e
|      | Payload: [128 bytes]
|      |
|      | (Usual GET transfer left out)
|      |
| ...  |
|      | (Notification of first block:)
|      |
| <-----+ Header: 2.05 0x4145af9c
| 2.05 | Token: 0xfb
|      | Block2: 0/1/128
|      | Observe: 62354
|      | ETag: 6f00f392
|      | Payload: [128 bytes]
|      |
| +-- - -> Header: 0x6000af9c
|      |
|      | (Retrieval of remaining blocks)
|      |
| +-----> Header: GET 0x41011637
| GET  | Token: 0xfc
|      | Uri-Path: status-icon
|      | Block2: 1/0/128
|      |
| <-----+ Header: 2.05 0x61451637
| 2.05 | Token: 0xfc
|      | Block2: 1/1/128
|      | ETag: 6f00f392
|      | Payload: [128 bytes]
|      |
| +-----> Header: GET 0x41011638
| GET  | Token: 0xfc
|      | Uri-Path: status-icon
|      | Block2: 2/0/128
|      |
| <-----+ Header: 2.05 0x61451638
| 2.05 | Token: 0xfc
|      | Block2: 2/0/128
|      | ETag: 6f00f392
|      | Payload: [53 bytes]

```

Figure 12: Observe sequence with block-wise response

(Note that the choice of token 0xfc in this examples is arbitrary; tokens are just shown in this example to illustrate that the requests

for additional blocks cannot make use of the token of the Observation relationship. As a general comment on tokens, there is no other mention of tokens in this document, as block-wise transfers handle tokens like any other CoAP exchange. As usual the client is free to choose tokens for each exchange as it likes.)

In the following example, the client also uses early negotiation to limit the block size to 64 bytes.

CLIENT	SERVER
+-----> GET	Header: GET 0x41011636 Token: 0xfb Uri-Path: status-icon Observe: (empty) Block2: 0/0/64
<-----+ 2.05	Header: 2.05 0x61451636 Token: 0xfb Block2: 0/1/64 Observe: 62350 ETag: 6f00f38e Max-Age: 60 Payload: [64 bytes]
	(Usual GET transfer left out)
...	(Notification of first block:)
<-----+ 2.05	Header: 2.05 0x4145af9c Token: 0xfb Block2: 0/1/64 Observe: 62354 ETag: 6f00f392 Payload: [64 bytes]
+- - ->	Header: 0x6000af9c (Retrieval of remaining blocks)
+-----> GET	Header: GET 0x41011637 Token: 0xfc Uri-Path: status-icon Block2: 1/0/64
<-----+ 2.05	Header: 2.05 0x61451637 Token: 0xfc Block2: 1/1/64

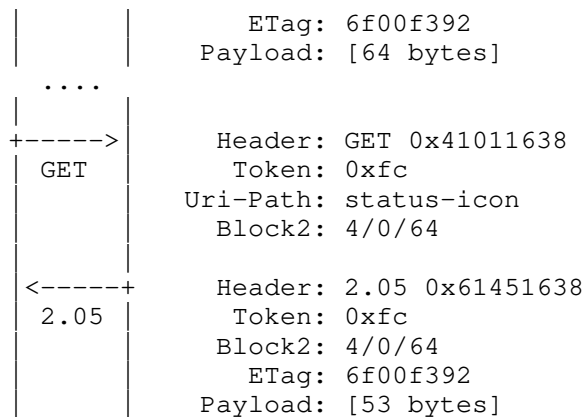


Figure 13: Observe sequence with early negotiation

4. The Size2 and Size1 Options

In many cases when transferring a large resource representation block by block, it is advantageous to know the total size early in the process. Some indication may be available from the maximum size estimate attribute "sz" provided in a resource description [RFC6690]. However, the size may vary dynamically, so a more up-to-date indication may be useful.

This specification defines two CoAP Options, Size1 for indicating the size of the representation transferred in requests, and Size2 for indicating the size of the representation transferred in responses. (Size1 has already been defined in Section 5.10.9 of [RFC7252] to provide "size information about the resource representation in a request", however that section only details the narrow case of indicating in 4.13 responses the maximum size of request payload that the server is able and willing to handle. The present specification provides details about its use as a request option as well.)

The Size2 Option may be used for two purposes:

- o in a request, to ask the server to provide a size estimate along with the usual response ("size request"). For this usage, the value MUST be set to 0.
- o in a response carrying a Block2 Option, to indicate the current estimate the server has of the total size of the resource representation, measured in bytes ("size indication").

Similarly, the Size1 Option may be used for two purposes:

- o in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation, measured in bytes ("size indication").
- o in a 4.13 response, to indicate the maximum size that would have been acceptable [RFC7252], measured in bytes.

Apart from conveying/asking for size information, the Size options have no other effect on the processing of the request or response. If the client wants to minimize the size of the payload in the resulting response, it should add a Block2 option to the request with a small block size (e.g., setting SZX=0).

The Size Options are "elective", i.e., a client MUST be prepared for the server to ignore the size estimate request. The Size Options MUST NOT occur more than once.

No.	C	U	N	R	Name	Format	Length	Default
60			x		Size1	uint	0-4	(none)
28			x		Size2	uint	0-4	(none)

Table 2: Size Option Numbers

Implementation Notes:

- o As a quality of implementation consideration, block-wise transfers for which the total size considerably exceeds the size of one block are expected to include size indications, whenever those can be provided without undue effort (preferably with the first block exchanged). If the size estimate does not change, the indication does not need to be repeated for every block.
- o The end of a block-wise transfer is governed by the M bits in the Block Options, not by exhausting the size estimates exchanged.
- o As usual for an option of type uint, the value 0 is best expressed as an empty option (0 bytes). There is no default value for either Size Option.
- o The Size Options are neither critical nor unsafe, and are marked as No-Cache-Key.

5. HTTP Mapping Considerations

In this subsection, we give some brief examples for the influence the Block options might have on intermediaries that map between CoAP and HTTP.

For mapping CoAP requests to HTTP, the intermediary may want to map the sequence of block-wise transfers into a single HTTP transfer. E.g., for a GET request, the intermediary could perform the HTTP request once the first block has been requested and could then fulfill all further block requests out of its cache. A constrained implementation may not be able to cache the entire object and may use a combination of TCP flow control and (in particular if timeouts occur) HTTP range requests to obtain the information necessary for the next block transfer at the right time.

For PUT or POST requests, historically there was more variation in how HTTP servers might implement ranges; recently, [RFC7233] has defined that Range header fields received with a request method other than GET are not to be interpreted. So, in general, the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer for these other methods within one HTTP request. If enough buffering is available, this request can be started when the last CoAP block is received. A constrained implementation may want to relieve its buffering by already starting to send the HTTP request at the time the first CoAP block is received; any HTTP 408 status code that indicates that the HTTP server became impatient with the resulting transfer can then be mapped into a CoAP 4.08 response code (similarly, 413 maps to 4.13).

For mapping HTTP to CoAP, the intermediary may want to map a single HTTP transfer into a sequence of block-wise transfers. If the HTTP client is too slow delivering a request body on a PUT or POST, the CoAP server might time out and return a 4.08 response code, which in turn maps well to an HTTP 408 status code (again, 4.13 maps to 413). HTTP range requests received on the HTTP side may be served out of a cache and/or mapped to GET requests that request a sequence of blocks overlapping the range.

(Note that, while the semantics of CoAP 4.08 and HTTP 408 differ, this difference is largely due to the different way the two protocols are mapped to transport. HTTP has an underlying TCP connection, which supplies connection state, so a HTTP 408 status code can immediately be used to indicate that a timeout occurred during transmitting a request through that active TCP connection. The CoAP 4.08 response code indicates one or more missing blocks, which may be due to timeouts or resource constraints; as there is no connection state, there is no way to deliver such a response immediately;

instead, it is delivered on the next block transfer. Still, HTTP 408 is probably the best mapping back to HTTP, as the timeout is the most likely cause for a CoAP 4.08. Note that there is no way to distinguish a timeout from a missing block for a server without creating additional state, the need for which we want to avoid.)

6. IANA Considerations

This draft adds the following option numbers to the CoAP Option Numbers registry of [RFC7252]:

Number	Name	Reference
23	Block2	[RFCXXXX]
27	Block1	[RFCXXXX]
28	Size2	[RFCXXXX]

Table 3: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [RFC7252]:

Code	Description	Reference
2.31	Continue	[RFCXXXX]
4.08	Request Entity Incomplete	[RFCXXXX]

Table 4: CoAP Response Codes

7. Security Considerations

Providing access to blocks within a resource may lead to surprising vulnerabilities. Where requests are not implemented atomically, an attacker may be able to exploit a race condition or confuse a server by inducing it to use a partially updated resource representation. Partial transfers may also make certain problematic data invisible to intrusion detection systems; it is RECOMMENDED that an intrusion detection system (IDS) that analyzes resource representations transferred by CoAP implement the Block options to gain access to entire resource representations. Still, approaches such as transferring even-numbered blocks on one path and odd-numbered blocks

on another path, or even transferring blocks multiple times with different content and obtaining a different interpretation of temporal order at the IDS than at the server, may prevent an IDS from seeing the whole picture. These kinds of attacks are well understood from IP fragmentation and TCP segmentation; CoAP does not add fundamentally new considerations.

Where access to a resource is only granted to clients making use of specific security associations, all blocks of that resource **MUST** be subject to the same security checks; it **MUST NOT** be possible for unprotected exchanges to influence blocks of an otherwise protected resource. As a related consideration, where object security is employed, PUT/POST should be implemented in the atomic fashion, unless the object security operation is performed on each access and the creation of unusable resources can be tolerated. Future end-to-end security mechanisms that may be added to CoAP itself may have related security considerations, this includes considerations about caching of blocks in clients and in proxies (see Section 2.10 and Section 5 for different strategies in performing this caching); these security considerations will need to be described in the specifications of those mechanisms.

A stateless server might be susceptible to an attack where the adversary sends a Block1 (e.g., PUT) block with a high block number: A naive implementation might exhaust its resources by creating a huge resource representation.

Misleading size indications may be used by an attacker to induce buffer overflows in poor implementations, for which the usual considerations apply.

7.1. Mitigating Resource Exhaustion Attacks

Certain block-wise requests may induce the server to create state, e.g. to create a snapshot for the block-wise GET of a fast-changing resource to enable consistent access to the same version of a resource for all blocks, or to create temporary resource representations that are collected until pressed into service by a final PUT or POST with the more bit unset. All mechanisms that induce a server to create state that cannot simply be cleaned up create opportunities for denial-of-service attacks. Servers **SHOULD** avoid being subject to resource exhaustion based on state created by untrusted sources. But even if this is done, the mitigation may cause a denial-of-service to a legitimate request when it is drowned out by other state-creating requests. Wherever possible, servers should therefore minimize the opportunities to create state for untrusted sources, e.g. by using stateless approaches.

Performing segmentation at the application layer is almost always better in this respect than at the transport layer or lower (IP fragmentation, adaptation layer fragmentation), for instance because there is application layer semantics that can be used for mitigation or because lower layers provide security associations that can prevent attacks. However, it is less common to apply timeouts and keepalive mechanisms at the application layer than at lower layers. Servers MAY want to clean up accumulated state by timing it out (cf. response code 4.08), and clients SHOULD be prepared to run block-wise transfers in an expedient way to minimize the likelihood of running into such a timeout.

7.2. Mitigating Amplification Attacks

[RFC7252] discusses the susceptibility of CoAP end-points for use in amplification attacks.

A CoAP server can reduce the amount of amplification it provides to an attacker by offering large resource representations only in relatively small blocks. With this, e.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

8.2. Informative References

- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <<http://www.rfc-editor.org/info/rfc4919>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<http://www.rfc-editor.org/info/rfc4944>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <<http://www.rfc-editor.org/info/rfc7233>>.

Acknowledgements

Much of the content of this draft is the result of discussions with the [RFC7252] authors, and via many CoRE WG discussions.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version. Esko Dijk reviewed a more recent version, leading to a number of further editorial improvements, a solution to the 4.13 ambiguity problem, and the section about combining Block and multicast. Markus Becker proposed getting rid of an ill-conceived default value for the Block2 and Block1 options. Peter Bigot

insisted on a more systematic coverage of the options and response code. Qin Wu provided a review for the IETF Operational directorate, and Goeran Selander commented on the security considerations.

Kepeng Li, Linyi Tian, and Barry Leiba wrote up an early version of the Size Option, which has informed this draft. Klaus Hartke wrote some of the text describing the interaction of Block2 with Observe. Matthias Kovatsch provided a number of significant simplifications of the protocol.

The IESG reviewers provided very useful comments. Spencer Dawkins even suggested new text. Mirja Kuehlewind and he insisted on being more explicit about the layering of block-wise transfers on top of the base protocol. Ben Campbell helped untangling some MUST/SHOULD soup. Comments by Alexey Melnikov, as well as the gen-art review by Jouni Korhonen and the ops-dir review by Qin Wu, caused further improvements to the text.

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Zach Shelby (editor)
ARM
150 Rose Orchard
San Jose, CA 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

CORE
Internet-Draft
Intended status: Standards Track
Expires: May 22, 2016

C. Bormann, Ed.
Universitaet Bremen TZI
S. Lemay
V. Solorzano Barboza
Zebra Technologies
H. Tschofenig
ARM Ltd.
November 19, 2015

A TCP and TLS Transport for the Constrained Application Protocol (CoAP)
draft-ietf-core-coap-tcp-tls-01

Abstract

The Hypertext Transfer Protocol (HTTP) was designed with TCP as the underlying transport protocol. The Constrained Application Protocol (CoAP), while inspired by HTTP, has been defined to make use of UDP instead of TCP. Therefore, reliable delivery and a simple congestion control and flow control mechanism are provided by the message layer of the CoAP protocol.

A number of environments benefit from the use of CoAP directly over a reliable byte stream such as TCP, which already provides these services. This document defines the use of CoAP over TCP as well as CoAP over TLS.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 22, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Constrained Application Protocol	3
4. Message Format	5
4.1. Discussion	6
5. Message Transmission	7
6. CoAP URI	7
6.1. coap+tcp URI scheme	7
6.2. coaps+tcp URI scheme	8
7. Security Considerations	8
8. IANA Considerations	8
8.1. Service Name and Port Number Registration	8
8.2. URI Schemes	9
8.3. ALPN Protocol ID	10
9. Acknowledgements	10
10. References	10
10.1. Normative References	10
10.2. Informative References	11
Authors' Addresses	12

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP can be used unimpeded -- UDP [RFC0768], or DTLS [RFC6347] over UDP; it is a good choice for transferring small amounts of data across networks that follow the IP architecture. Some CoAP deployments, however, may have to integrate well with existing enterprise infrastructure, where the use of UDP-based protocols may not be well-received or may even be blocked by firewalls. Middleboxes that are unaware of CoAP usage

for IoT can make the use of UDP brittle, resulting in lost or malformed packets.

Where NATs are still present, CoAP over TCP can also help with their traversal. NATs often calculate expiration timers based on the transport layer protocol being used by application protocols. Many NATs are built around the assumption that a transport layer protocol such as TCP gives them additional information about the session life cycle and keep TCP-based NAT bindings around for a longer period. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter, as research confirms [HomeGateway].

Some environments may also benefit from the more sophisticated congestion control capabilities provided by many TCP implementations. (Note that there is ongoing work to add more elaborate congestion control to CoAP as well, see [I-D.bormann-core-cocoa].)

Finally, CoAP may be integrated into a Web environment where the front-end uses CoAP from IoT devices to a cloud infrastructure but the CoAP messages are then transported in TCP between the back-end services. A TCP-to-UDP gateway can be used at the cloud boundary to talk to the UDP-based IoT.

To make IoT devices work smoothly in these demanding environments, CoAP needs to make use of a different transport protocol, namely TCP [RFC0793], in some situations secured by TLS [RFC5246].

The present document describes a shim header that conveys length information about each CoAP message. Modifications to CoAP beyond the replacement of the message layer (e.g., to introduce further optimizations) are intentionally avoided.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Constrained Application Protocol

The interaction model of CoAP over TCP is very similar to the one for CoAP over UDP, with the key difference that using TCP voids the need to provide certain transport layer protocol features, such as reliable delivery, fragmentation and reassembly, as well as congestion control, at the CoAP level. The protocol stack is illustrated in Figure 1 (derived from [RFC7252], Figure 1).

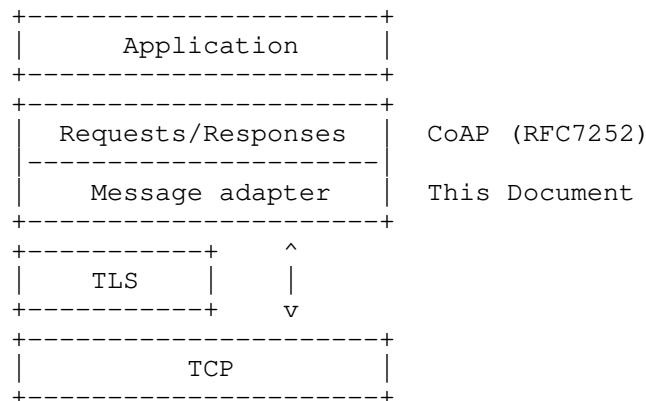


Figure 1: The CoAP over TLS/TCP Protocol Stack

Since TCP offers reliable delivery, there is no need to offer a redundant acknowledgement at the CoAP messaging layer.

Since there is no need to carry around acknowledgement semantics, messages do not require a message type; no message layer acknowledgement is expected or even possible. Because something needs to be put into the two bits indicating the message type, we put the bits for a Non-Confirmable message (NON) into the header. By the nature of TCP, messages are always transmitted reliably over TCP. Figure 2 (derived from [RFC7252], Figure 3) shows this message exchange graphically. A UDP-to-TCP gateway will therefore discard all empty messages, such as empty ACKs (after operating on them at the message layer), and re-pack the contents of all non-empty CON, NON, or ACK messages (i.e., those ACK messages that have a piggy-backed response) into untyped messages (that happen to look like NON messages).

Similarly, there is no need to detect duplicate delivery of a message. In UDP CoAP, the Message ID is used for relating acknowledgements to Confirmable messages as well as for duplicate detection. Since the Message ID thus is not meaningful over TCP, it is elided (as indicated by the dashes in Figure 2).

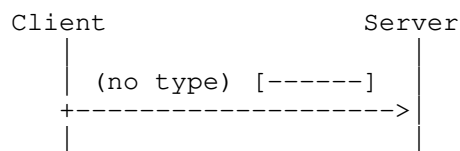


Figure 2: Untyped Message Transmission over TCP.

A response is sent back as defined in [RFC7252], as illustrated in Figure 3 (derived from [RFC7252], Figure 6).

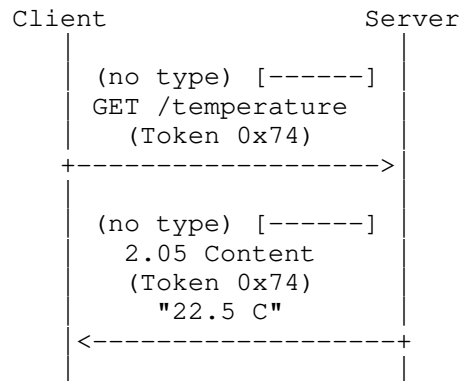


Figure 3

4. Message Format

The CoAP message format defined in [RFC7252], as shown in Figure 4, relies on the datagram transport (UDP, or DTLS over UDP) for keeping the individual messages separate.

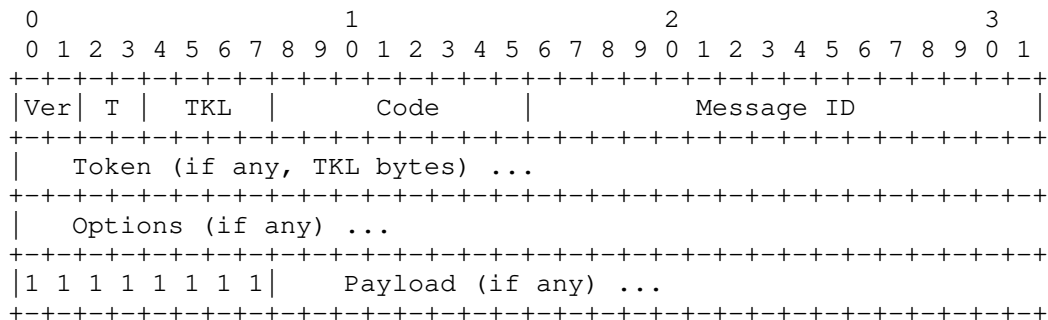


Figure 4: RFC 7252 defined CoAP Message Format.

In a stream oriented transport protocol such as TCP, a form of message delimitation is needed. For this purpose, CoAP over TCP introduces a length field. Figure 5 shows a 2-byte shim header carrying length information prepended to the CoAP message header.

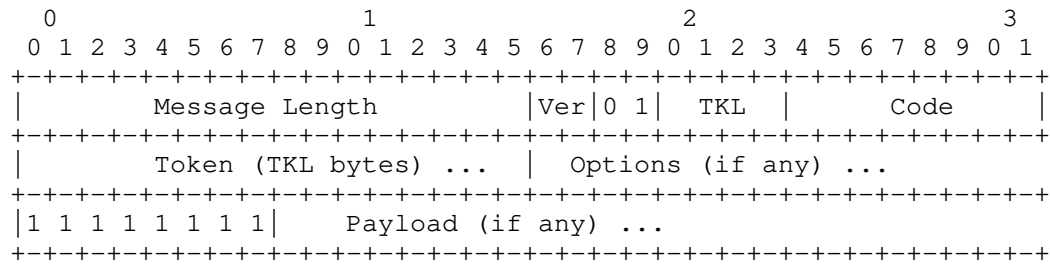


Figure 5: CoAP Header with prepended Shim Header.

The 'Message Length' field is a 16-bit unsigned integer in network byte order. It provides the length of the subsequent CoAP message (including the CoAP header but excluding this message length field) in bytes (so its minimum value is 2). The Message ID and message type are meaningless and thus elided (what would have been the message type field is always filled with what would be the code for NON (01)).

The semantics of the other CoAP header fields are left unchanged.

4.1. Discussion

One observation is that, over a reliable byte stream transport, the message size limitations defined in Section 4.6 of [RFC7252] are no longer strictly necessary. Consenting [[how: There is currently no defined way to arrive at this consent. --cabo]] implementations may want to interchange messages with payload sizes larger than 1024 bytes, potentially also obviating the need for the Block protocol [I-D.ietf-core-block]. It must be noted that entirely getting rid of the block protocol is not a generally applicable solution, as:

- o a UDP-to-TCP gateway may simply not have the context to convert a message with a Block option into the equivalent exchange without any use of a Block option;
- o large messages might also cause undesired head-of-line blocking;
- o the 2-byte message length field causes another, larger upper bound to the message length.

The general assumption is therefore that the block protocol will continue to be used over TCP, even if TCP-based applications occasionally do exchange messages with payload sizes larger than desirable in UDP.

5. Message Transmission

As CoAP exchanges messages asynchronously over the TCP connection, the client can send multiple requests without waiting for responses. For this reason, and due to the nature of TCP, responses are returned during the same TCP connection as the request. In the event that the connection gets terminated, all requests that have not yet elicited a response are implicitly canceled; clients may transmit the request again once a connection is reestablished.

Furthermore, since TCP is bidirectional, requests can be sent from both the connecting host and the endpoint that accepted the connection. In other words, the question who initiated the TCP connection has no bearing on the meaning of the CoAP terms client and server.

6. CoAP URI

CoAP [RFC7252] defines the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. RFC 7252 defines these resources for use with CoAP over UDP.

The present specification introduces two new URI schemes, namely "coap+tcp" and "coaps+tcp". The rules from Section 6 of [RFC7252] apply to these two new URI schemes.

[RFC7252], Section 8 (Multicast CoAP), does not apply to the URI schemes defined in the present specification.

Resources made available via one of the "coap+tcp" or "coaps+tcp" schemes have no shared identity with the other scheme or with the "coap" or "coaps" scheme, even if their resource identifiers indicate the same authority (the same host listening to the same port). The schemes constitute distinct namespaces and, in combination with the authority, are considered to be distinct origin servers.

6.1. coap+tcp URI scheme

```
coap-tcp-URI = "coap+tcp:" "/" host [ ":" port ] path-abempty
               [ "?" query ]
```

The semantics defined in [RFC7252], Section 6.1, apply to this URI scheme, with the following changes:

- o The port subcomponent indicates the TCP port at which the CoAP server is located. (If it is empty or not given, then the default port 5683 is assumed, as with UDP.)

6.2. coaps+tcp URI scheme

```
coaps-tcp-URI = "coaps+tcp:" "://" host [ ":" port ] path-abempty  
                [ "?" query ]
```

The semantics defined in [RFC7252], Section 6.2, apply to this URI scheme, with the following changes:

- o The port subcomponent indicates the TCP port at which the TLS server for the CoAP server is located. If it is empty or not given, then the default port 443 is assumed (this is different from the default port for "coaps", i.e., CoAP over DTLS over UDP).
- o When CoAP is exchanged over TLS port 443 then the "TLS Application Layer Protocol Negotiation Extension" [RFC7301] MUST be used to allow demultiplexing at the server-side unless out-of-band information ensures that the client only interacts with a server that is able to demultiplex CoAP messages over port 443. This would, for example, be true for many IoT deployments where clients are pre-configured to only ever talk with specific servers.
[[alwaysalpn: Shouldn't we simply always require ALPN? The protocol should not be defined in such a way that it depends on some undefined pre-configuration mechanism. --cabo]]

7. Security Considerations

This document defines how to convey CoAP over TCP and TLS. It does not introduce new vulnerabilities beyond those described already in the CoAP specification. CoAP [RFC7252] makes use of DTLS 1.2 and this specification consequently uses TLS 1.2 [RFC5246]. CoAP MUST NOT be used with older versions of TLS. Guidelines for use of cipher suites and TLS extensions can be found in [I-D.ietf-dice-profile].

8. IANA Considerations

8.1. Service Name and Port Number Registration

IANA is requested to assign the port number 5683 and the service name "coap+tcp", in accordance with [RFC6335].

Service Name.
coap+tcp

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

Constrained Application Protocol (CoAP)

Reference.

[RFCthis]

Port Number.

5683

Similarly, IANA is requested to assign the service name "coaps+tcp", in accordance with [RFC6335]. However, no separate port number is used for "coaps" over TCP; instead, the ALPN protocol ID defined in Section 8.3 is used over port 443.

Service Name.

coaps+tcp

Transport Protocol.

tcp

Assignee.

IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

Constrained Application Protocol (CoAP)

Reference.

[RFC7301], [RFCthis]

Port Number.

443 (see also Section 8.3 of [RFCthis])

8.2. URI Schemes

This document registers two new URI schemes, namely "coap+tcp" and "coaps+tcp", for the use of CoAP over TCP and for CoAP over TLS over TCP, respectively. The "coap+tcp" and "coaps+tcp" URI schemes can thus be compared to the "http" and "https" URI schemes.

The syntax of the "coap" and "coaps" URI schemes is specified in Section 6 of [RFC7252] and the present document re-uses their

semantics for "coap+tcp" and "coaps+tcp", respectively, with the exception that TCP, or TLS over TCP is used as a transport protocol.

IANA is requested to add these new URI schemes to the registry established with [RFC7595].

8.3. ALPN Protocol ID

IANA is requested to assign the following value in the registry "Application Layer Protocol Negotiation (ALPN) Protocol IDs" created by [RFC7301]:

Protocol:
CoAP

Identification Sequence:
0x63 0x6f 0x61 0x70 ("coap")

Reference:
[RFCthis]

9. Acknowledgements

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Michael Koster, Matthias Kovatsch, Szymon Sasin, Andrew Summers, and Zach Shelby for their feedback.

10. References

10.1. Normative References

- [I-D.ietf-dice-profile] Tschofenig, H. and T. Fossati, "TLS/DTLS Profiles for the Internet of Things", draft-ietf-dice-profile-17 (work in progress), October 2015.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<http://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<http://www.rfc-editor.org/info/rfc7301>>.
- [RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<http://www.rfc-editor.org/info/rfc7595>>.

10.2. Informative References

- [HomeGateway] Eggert, L., "An experimental study of home gateway characteristics", Proceedings of the 10th annual conference on Internet measurement, 2010.
- [I-D.bormann-core-cocoa] Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-bormann-core-cocoa-03 (work in progress), October 2015.
- [I-D.ietf-core-block] Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<http://www.rfc-editor.org/info/rfc768>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<http://www.rfc-editor.org/info/rfc6335>>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.

Authors' Addresses

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Simon Lemay
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: slemay@zebra.com

Valik Solorzano Barboza
Zebra Technologies
820 W. Jackson Blvd. suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: vsolorzanobarboza@zebra.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
Great Britain

Email: Hannes.tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

CORE
Internet-Draft
Updates: 7641, 7959 (if approved)
Intended status: Standards Track
Expires: June 21, 2018

C. Bormann
Universitaet Bremen TZI
S. Lemay
Zebra Technologies
H. Tschofenig
ARM Ltd.
K. Hartke
Universitaet Bremen TZI
B. Silverajan
Tampere University of Technology
B. Raymor, Ed.
December 18, 2017

CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets
draft-ietf-core-coap-tcp-tls-11

Abstract

The Constrained Application Protocol (CoAP), although inspired by HTTP, was designed to use UDP instead of TCP. The message layer of the CoAP over UDP protocol includes support for reliable delivery, simple congestion control, and flow control.

Some environments benefit from the availability of CoAP carried over reliable transports such as TCP or TLS. This document outlines the changes required to use CoAP over TCP, TLS, and WebSockets transports. It also formally updates RFC 7641 for use with these transports and RFC 7959 to enable the use of larger messages over a reliable transport.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 21, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Conventions and Terminology	6
3. CoAP over TCP	7
3.1. Messaging Model	7
3.2. Message Format	8
3.3. Message Transmission	10
3.4. Connection Health	11
4. CoAP over WebSockets	12
4.1. Opening Handshake	13
4.2. Message Format	14
4.3. Message Transmission	15
4.4. Connection Health	15
5. Signaling	15
5.1. Signaling Codes	16
5.2. Signaling Option Numbers	16
5.3. Capabilities and Settings Messages (CSM)	16
5.4. Ping and Pong Messages	19
5.5. Release Messages	20
5.6. Abort Messages	21
5.7. Signaling examples	22
6. Block-wise Transfer and Reliable Transports	23
6.1. Example: GET with BERT Blocks	24
6.2. Example: PUT with BERT Blocks	25
7. Observing Resources over Reliable Transports	25
7.1. Notifications and Reordering	26
7.2. Transmission and Acknowledgements	26
7.3. Freshness	26
7.4. Cancellation	27
8. CoAP over Reliable Transport URIs	27
8.1. coap+tcp URI scheme	28
8.2. coaps+tcp URI scheme	28

8.3.	coap+ws URI scheme	29
8.4.	coaps+ws URI scheme	30
8.5.	Uri-Host and Uri-Port Options	31
8.6.	Decomposing URIs into Options	31
8.7.	Composing URIs from Options	32
9.	Securing CoAP	32
9.1.	TLS binding for CoAP over TCP	33
9.2.	TLS usage for CoAP over WebSockets	34
10.	Security Considerations	34
10.1.	Signaling Messages	34
11.	IANA Considerations	34
11.1.	Signaling Codes	34
11.2.	CoAP Signaling Option Numbers Registry	35
11.3.	Service Name and Port Number Registration	36
11.4.	Secure Service Name and Port Number Registration	37
11.5.	URI Scheme Registration	38
11.6.	Well-Known URI Suffix Registration	40
11.7.	ALPN Protocol Identifier	40
11.8.	WebSocket Subprotocol Registration	40
11.9.	CoAP Option Numbers Registry	41
12.	References	41
12.1.	Normative References	41
12.2.	Informative References	43
	Appendix A. CoAP over WebSocket Examples	45
	Appendix B. Change Log	48
	B.1. Since draft-ietf-core-coap-tcp-tls-02	48
	B.2. Since draft-ietf-core-coap-tcp-tls-03	48
	B.3. Since draft-ietf-core-coap-tcp-tls-04	48
	B.4. Since draft-ietf-core-coap-tcp-tls-05	48
	B.5. Since draft-ietf-core-coap-tcp-tls-06	49
	B.6. Since draft-ietf-core-coap-tcp-tls-07	49
	Acknowledgements	49
	Contributors	49
	Authors' Addresses	50

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP [RFC0768] can be used unimpeded, as can the Datagram Transport Layer Security protocol (DTLS [RFC6347]) over UDP. The use of CoAP over UDP is focused on simplicity, has a low code footprint, and a small over-the-wire message size.

The primary reason for introducing CoAP over TCP [RFC0793] and TLS [RFC5246] is that some networks do not forward UDP packets. Complete blocking of UDP happens in between about 2% and 4% of terrestrial access networks, according to [EK2016]. UDP impairment is especially

concentrated in enterprise networks and networks in geographic regions with otherwise challenged connectivity. Some networks also rate-limit UDP traffic, as reported in [BK2015] and deployment investigations related to the standardization of QUIC revealed numbers around 0.3 % [SW2016].

The introduction of CoAP over TCP also leads to some additional effects that may be desirable in a specific deployment:

- o Where NATs are present along the communication path, CoAP over TCP leads to different NAT traversal behavior than CoAP over UDP. NATs often calculate expiration timers based on the transport layer protocol being used by application protocols. Many NATs maintain TCP-based NAT bindings for longer periods based on the assumption that a transport layer protocol, such as TCP, offers additional information about the session lifecycle. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter [HomeGateway]. According to [HomeGateway] the mean for TCP and UDP NAT binding timeouts is 386 minutes (TCP) and 160 seconds (UDP). Shorter timeout values require keepalive messages to be sent more frequently. Hence, the use of CoAP over TCP requires less frequent transmission of keep-alive messages.
- o TCP utilizes more sophisticated congestion and flow control mechanisms than the default mechanisms provided by CoAP over UDP, which is useful for the transfer of larger payloads. (Work is, however, ongoing to add advanced congestion control to CoAP over UDP as well, see [I-D.ietf-core-cocoa].)

Note that the use of CoAP over UDP (and CoAP over DTLS over UDP) is still the recommended transport for use in constrained node networks, particularly when used in concert with blockwise transfer. CoAP over TCP is applicable for those cases where the networking infrastructure leaves no other choice. The use of CoAP over TCP leads to a larger code size, more roundtrips, increased RAM requirements and larger packet sizes. Developers implementing CoAP over TCP are encouraged to consult [I-D.gomez-lwig-tcp-constrained-node-networks] for guidance on low-footprint TCP implementations for IoT devices.

Standards based on CoAP such as Lightweight Machine to Machine [LWM2M] currently use CoAP over UDP as a transport; adding support for CoAP over TCP enables them to address the issues above for specific deployments and to protect investments in existing CoAP implementations and deployments.

Although HTTP/2 could also potentially address the need for enterprise firewall traversal, there would be additional costs and

delays introduced by such a transition from CoAP to HTTP/2. Currently, there are also fewer HTTP/2 implementations available for constrained devices in comparison to CoAP. Since CoAP also support group communication using IP layer multicast and unreliable communication IoT devices would have to support HTTP/2 in addition to CoAP.

Furthermore, CoAP may be integrated into a Web environment where the front-end uses CoAP over UDP from IoT devices to a cloud infrastructure and then CoAP over TCP between the back-end services. A TCP-to-UDP gateway can be used at the cloud boundary to communicate with the UDP-based IoT device.

Finally, CoAP applications running inside a web browser may be without access to connectivity other than HTTP. In this case, the WebSocket protocol [RFC6455] may be used to transport CoAP requests and responses, as opposed to cross-proxying them via HTTP to an HTTP-to-CoAP cross-proxy. This preserves the functionality of CoAP without translation, in particular the Observe mechanism [RFC7641].

To address the above-mentioned deployment requirements, this document defines how to transport CoAP over TCP, CoAP over TLS, and CoAP over WebSockets. For these cases, the reliability offered by the transport protocol subsumes the reliability functions of the message layer used for CoAP over UDP. (Note that both for a reliable transport and the CoAP over UDP message layer, the reliability offered is per transport hop: where proxies -- see Sections 5.7 and 10 of [RFC7252] -- are involved, that layer's reliability function does not extend end-to-end.) Figure 1 illustrates the layering:

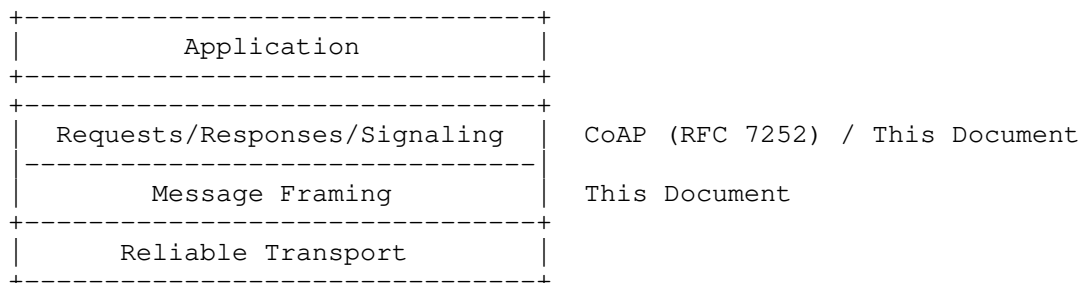


Figure 1: Layering of CoAP over Reliable Transports

This document specifies how to access resources using CoAP requests and responses over the TCP, TLS and WebSocket protocols. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either by communicating CoAP directly with a CoAP server accessible over a TCP, TLS or WebSocket connection or via a CoAP

intermediary that proxies CoAP requests and responses between different transports, such as between WebSockets and UDP.

Section 7 updates the "Observing Resources in the Constrained Application Protocol" [RFC7641] specification for use with CoAP over reliable transports. [RFC7641] is an extension to the CoAP protocol that enables CoAP clients to "observe" a resource on a CoAP server. (The CoAP client retrieves a representation of a resource and registers to be notified by the CoAP server when the representation is updated.)

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455], [RFC7252], [RFC7641], and [RFC7959].

The term "reliable transport" is used only to refer to transport protocols, such as TCP, which provide reliable and ordered delivery of a byte-stream.

Block-wise Extension for Reliable Transport (BERT):

BERT extends [RFC7959] to enable the use of larger messages over a reliable transport.

BERT Option:

A Block1 or Block2 option that includes an SZX value of 7.

BERT Block:

The payload of a CoAP message that is affected by a BERT Option in descriptive usage (see Section 2.1 of [RFC7959]).

Transport Connection:

Underlying reliable byte stream connection, as directly provided by TCP, or indirectly via TLS or WebSockets.

Connection:

Transport Connection, unless explicitly qualified otherwise.

Connection Initiator:

The peer that opens a Transport Connection, i.e., the TCP active opener, TLS client, or WebSocket client.

Connection Acceptor:

The peer that accepts the Transport Connection opened by the other peer, i.e., the TCP passive opener, TLS server, or WebSocket server.

3. CoAP over TCP

The request/response interaction model of CoAP over TCP is the same as CoAP over UDP. The primary differences are in the message layer. The message layer of CoAP over UDP supports optional reliability by defining four types of messages: Confirmable, Non-confirmable, Acknowledgement, and Reset. In addition, messages include a Message ID to relate Acknowledgments to Confirmable messages and to detect duplicate messages.

The management of the transport connections is left to the application, i.e., the present specification does not describe how an application decides to open a connection or to re-open another one in the presence of failures (or what it would deem to be a failure, see also Section 5.4). In particular, the Connection Initiator need not be the client of the first request placed on the connection. Some implementations will want to implement a dynamic connection management similar to the one described in Section 6 of [RFC7230] for HTTP, opening a connection when the first client request is ready to be sent and reusing that for further messages for a while, until no message is sent for a certain time and no requests are outstanding (possibly with a configurable idle time) and a release process is started (Section 5.5). In implementations of this kind, connection releases or aborts may not be indicated as errors to the application but may simply be handled by automatic reconnection once the need arises again. Other implementations may be based on configured connections that are kept open continuously and lead to management system notifications on release or abort. The protocol defined in the present specification is intended to work with either model (or other, application-specific connection management models).

3.1. Messaging Model

Conceptually, CoAP over TCP replaces most of the message layer of CoAP over UDP with a framing mechanism on top of the byte-stream provided by TCP/TLS, conveying the length information for each message that on datagram transports is provided by the UDP/DTLS datagram layer.

TCP ensures reliable message transmission, so the message layer of CoAP over TCP is not required to support acknowledgements or to detect duplicate messages. As a result, both the Type and Message ID

fields are no longer required and are removed from the CoAP over TCP message format.

Figure 2 illustrates the difference between CoAP over UDP and CoAP over reliable transport. The removed Type and Message ID fields are indicated by dashes.

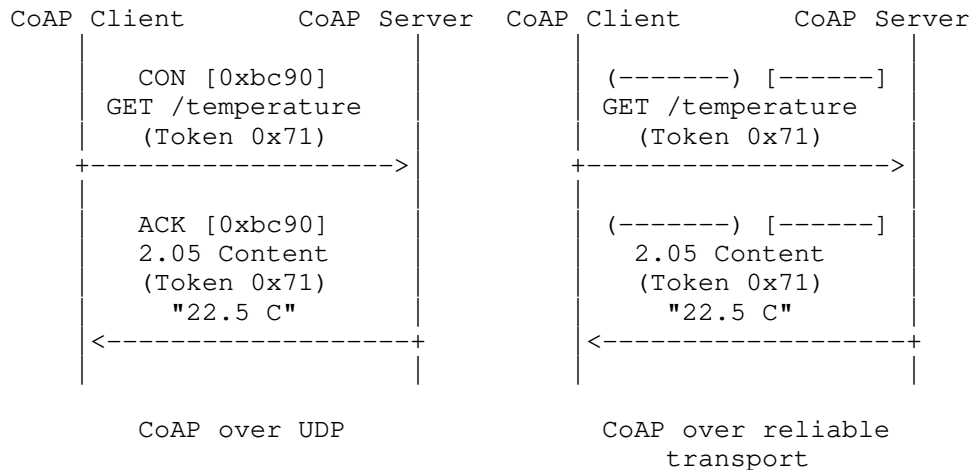


Figure 2: Comparison between CoAP over unreliable and reliable transport

3.2. Message Format

The CoAP message format defined in [RFC7252], as shown in Figure 3, relies on the datagram transport (UDP, or DTLS over UDP) for keeping the individual messages separate and for providing length information.

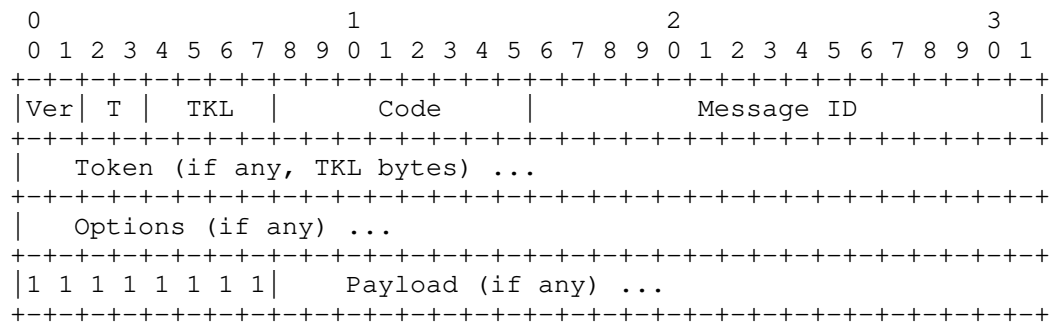


Figure 3: RFC 7252 defined CoAP Message Format

The CoAP over TCP message format is very similar to the format specified for CoAP over UDP. The differences are as follows:

- o Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. The Type (T) and Message ID fields in the CoAP message header are elided.
- o The Version (Vers) field is elided as well. In contrast to the message format of CoAP over UDP, the message format for CoAP over TCP does not include a version number. CoAP is defined in [RFC7252] with a version number of 1. At this time, there is no known reason to support version numbers different from 1. If version negotiation needs to be addressed in the future, then Capabilities and Settings Messages (CSM see Section 5.3) have been specifically designed to enable such a potential feature.
- o In a stream oriented transport protocol such as TCP, a form of message delimitation is needed. For this purpose, CoAP over TCP introduces a length field with variable size. Figure 4 shows the adjusted CoAP message format with a modified structure for the fixed header (first 4 bytes of the CoAP over UDP header), which includes the length information of variable size.

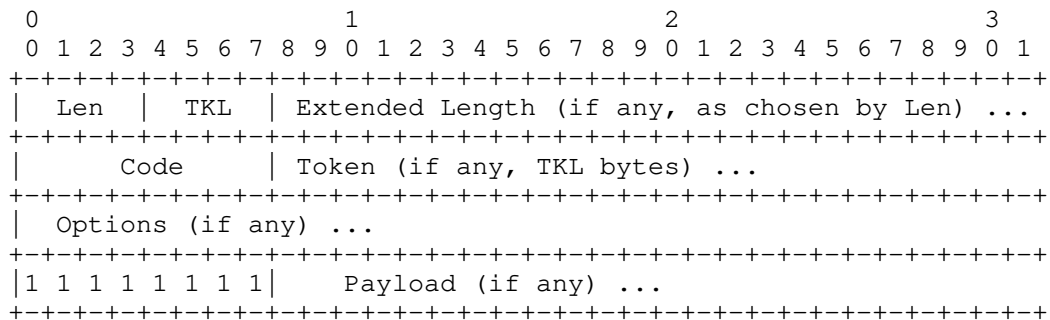


Figure 4: CoAP frame for reliable transports

Length (Len): 4-bit unsigned integer. A value between 0 and 12 inclusive indicates the length of the message in bytes starting with the first bit of the Options field. Three values are reserved for special constructs:

- 13: An 8-bit unsigned integer (Extended Length) follows the initial byte and indicates the length of options/payload minus 13.

14: A 16-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 269.

15: A 32-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 65805.

The encoding of the Length field is modeled after the Option Length field of the CoAP Options (see Section 3.1 of [RFC7252]).

For simplicity, a Payload Marker (0xFF) is shown in Figure 4; the Payload Marker indicates the start of the optional payload and is absent for zero-length payloads (see Section 3 of [RFC7252]). (If present, the Payload Marker is included in the message length, which counts from the start of the Options field to the end of the Payload field.)

For example: A CoAP message just containing a 2.03 code with the token 7f and no options or payload is encoded as shown in Figure 5.

```

      0              1              2
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           0x01           |           0x43           |           0x7f           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

Len   =    0 -----> 0x01
TKL   =    1 ____/
Code  =  2.03      --> 0x43
Token =              0x7f

```

Figure 5: CoAP message with no options or payload

The semantics of the other CoAP header fields are left unchanged.

3.3. Message Transmission

Once a transport connection is established, each endpoint MUST send a Capabilities and Settings message (CSM, see Section 5.3) as their first message on the connection. This message establishes the initial settings and capabilities for the endpoint, such as maximum message size or support for block-wise transfers. The absence of options in the CSM indicates that base values are assumed.

To avoid a deadlock, the Connection Initiator MUST NOT wait for the Connection Acceptor to send its initial CSM message before sending its own initial CSM message. Conversely, the Connection Acceptor MAY

wait for the Connection Initiator to send its initial CSM message before sending its own initial CSM message.

To avoid unnecessary latency, a Connection Initiator MAY send additional messages after its initial CSM without waiting to receive the Connection Acceptor's CSM; however, it is important to note that the Connection Acceptor's CSM might indicate capabilities that impact how the initiator is expected to communicate with the acceptor. For example, the acceptor CSM could indicate a Max-Message-Size option (see Section 5.3.1) that is smaller than the base value (1152) in order to limit both buffering requirements and head-of-line blocking.

Endpoints MUST treat a missing or invalid CSM as a connection error and abort the connection (see Section 5.6).

CoAP requests and responses are exchanged asynchronously over the transport connection. A CoAP client can send multiple requests without waiting for a response and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Concurrent requests are differentiated by their Token, which is scoped locally to the connection.

The transport connection is bi-directional, so requests can be sent both by the entity that established the connection (Connection Initiator) and the remote host (Connection Acceptor). If one side does not implement a CoAP server, an error response MUST be returned for all CoAP requests from the other side. The simplest approach is to always return 5.01 (Not Implemented). A more elaborate mock server could also return 4.xx responses such as 4.04 (Not Found) or 4.02 (Bad Option) where appropriate.

Retransmission and deduplication of messages is provided by the TCP protocol.

3.4. Connection Health

Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient. This provides a basic keep-alive function that can refresh NAT bindings.

If a CoAP client does not receive any response for some time after sending a CoAP request (or, similarly, when a client observes a resource and it does not receive any notification for some time), it can send a CoAP Ping Signaling message (see Section 5.4) to test the transport connection and verify that the CoAP server is responsive.

When the underlying transport connection is closed or reset, the signaling state and any observation state (see Section 7.4) associated with the connection are removed. In flight messages may or may not be lost.

4. CoAP over WebSockets

CoAP over WebSockets is intentionally similar to CoAP over TCP; therefore, this section only specifies the differences between the transports.

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client retrieving or updating a CoAP resource located on a CoAP server that exposes a WebSocket endpoint (see Figure 6). The CoAP client acts as the WebSocket client, establishes a WebSocket connection, and sends a CoAP request, to which the CoAP server returns a CoAP response. The WebSocket connection can be used for any number of requests.

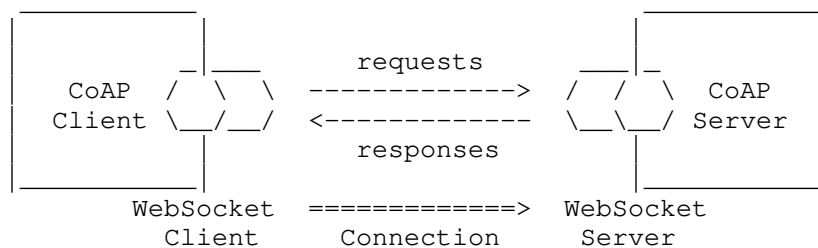


Figure 6: CoAP Client (WebSocket client) accesses CoAP Server (WebSocket server)

The challenge with this configuration is how to identify a resource in the namespace of the CoAP server. When the WebSocket protocol is used by a dedicated client directly (i.e., not from a web page through a web browser), the client can connect to any WebSocket endpoint. Section 8.3 and Section 8.4 define new URI schemes that enable the client to identify both a WebSocket endpoint and the path and query of the CoAP resource within that endpoint.

Another possible configuration is to set up a CoAP forward proxy at the WebSocket endpoint. Depending on what transports are available to the proxy, it could forward the request to a CoAP server with a CoAP UDP endpoint (Figure 7), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The CoAP client specifies the resource to be updated or retrieved in the Proxy-Uri Option.

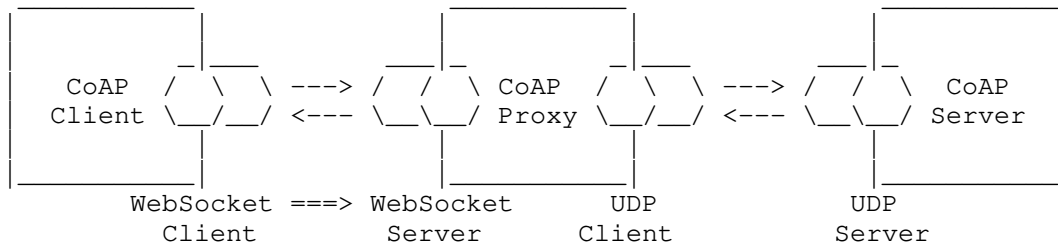


Figure 7: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

A third possible configuration is a CoAP server running inside a web browser (Figure 8). The web browser initially connects to a WebSocket endpoint and is then reachable through the WebSocket server. When no connection exists, the CoAP server is unreachable. Because the WebSocket server is the only way to reach the CoAP server, the CoAP proxy should be a reverse-proxy.

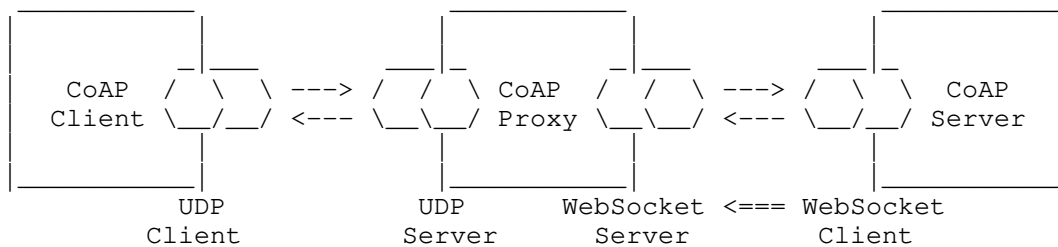


Figure 8: CoAP Client (UDP client) accesses CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

Further configurations are possible, including those where a WebSocket connection is established through an HTTP proxy.

4.1. Opening Handshake

Before CoAP requests and responses are exchanged, a WebSocket connection is established as defined in Section 4 of [RFC6455]. Figure 9 shows an example.

The WebSocket client MUST include the subprotocol name "coap" in the list of protocols, which indicates support for the protocol defined in this document.

The WebSocket client includes the hostname of the WebSocket server in the Host header field of its handshake as per [RFC6455]. The Host

Requests and response messages can be fragmented as specified in Section 5.4 of [RFC6455], though typically they are sent unfragmented as they tend to be small and fully buffered before transmission. The WebSocket protocol does not provide means for multiplexing. If it is not desirable for a large message to monopolize the connection, requests and responses can be transferred in a block-wise fashion as defined in [RFC7959].

4.3. Message Transmission

As with CoAP over TCP, each endpoint MUST send a Capabilities and Settings message (CSM see Section 5.3) as their first message on the WebSocket connection.

CoAP requests and responses are exchanged asynchronously over the WebSocket connection. A CoAP client can send multiple requests without waiting for a response and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Concurrent requests are differentiated by their Token, which is scoped locally to the connection.

The connection is bi-directional, so requests can be sent both by the entity that established the connection and the remote host.

As with CoAP over TCP, retransmission and deduplication of messages is provided by the WebSocket protocol. CoAP over WebSockets therefore does not make a distinction between Confirmable or Non-Confirmable messages, and does not provide Acknowledgement or Reset messages.

4.4. Connection Health

As with CoAP over TCP, a CoAP client can test the health of the CoAP over WebSocket connection by sending a CoAP Ping Signaling message (Section 5.4). WebSocket Ping and unsolicited Pong frames (Section 5.5 of [RFC6455]) SHOULD NOT be used to ensure that redundant maintenance traffic is not transmitted.

5. Signaling

Signaling messages are specifically introduced only for CoAP over reliable transports to allow peers to:

- o Learn related characteristics, such as maximum message size for the connection
- o Shut down the connection in an orderly fashion

- o Provide diagnostic information when terminating a connection in response to a serious error condition

Signaling is a third basic kind of message in CoAP, after requests and responses. Signaling messages share a common structure with the existing CoAP messages. There is a code, a token, options, and an optional payload.

(See Section 3 of [RFC7252] for the overall structure of the message format, option format, and option value format.)

5.1. Signaling Codes

A code in the 7.00-7.31 range indicates a Signaling message. Values in this range are assigned by the "CoAP Signaling Codes" sub-registry (see Section 11.1).

For each message, there is a sender and a peer receiving the message.

Payloads in Signaling messages are diagnostic payloads as defined in Section 5.5.2 of [RFC7252]), unless otherwise defined by a Signaling message option.

5.2. Signaling Option Numbers

Option numbers for Signaling messages are specific to the message code. They do not share the number space with CoAP options for request/response messages or with Signaling messages using other codes.

Option numbers are assigned by the "CoAP Signaling Option Numbers" sub-registry (see Section 11.2).

Signaling options are elective or critical as defined in Section 5.4.1 of [RFC7252]. If a Signaling option is critical and not understood by the receiver, it MUST abort the connection (see Section 5.6). If the option is understood but cannot be processed, the option documents the behavior.

5.3. Capabilities and Settings Messages (CSM)

Capabilities and Settings messages (CSM) are used for two purposes:

- o Each capability option indicates one capability of the sender to the recipient.
- o Each setting option indicates a setting that will be applied by the sender.

One CSM MUST be sent by each endpoint at the start of the transport connection. Further CSM MAY be sent at any other time by either endpoint over the lifetime of the connection.

Both capability and setting options are cumulative. A CSM does not invalidate a previously sent capability indication or setting even if it is not repeated. A capability message without any option is a no-operation (and can be used as such). An option that is sent might override a previous value for the same option. The option defines how to handle this case if needed.

Base values are listed below for CSM Options. These are the values for the capability and setting before any Capabilities and Settings messages send a modified value.

These are not default values for the option, as defined in Section 5.4.4 in [RFC7252]. Default values apply on a per-message basis and thus reset when the value is not present in a given Capabilities and Settings message.

Capabilities and Settings messages are indicated by the 7.01 code (CSM).

5.3.1. Max-Message-Size Capability Option

The sender can use the elective Max-Message-Size Option to indicate the maximum size of a message in bytes that it can receive. The message size indicated includes the entire message, starting from the first byte of the message header and ending at the end of the message payload.

(Note that there is no relationship of the message size to the overall request or response body size that may be achievable in block-wise transfer. For example, the exchange depicted further down in Figure 13 can be performed if the CoAP client indicates a value of around 6000 bytes for the Max-Message-Size option, even though the total body size transferred to the client is $3072 + 5120 + 4711 = 12903$ bytes.)

#	C	R	Applies to	Name	Format	Length	Base Value
2			CSM	Max-Message-Size	uint	0-4	1152

C=Critical, R=Repeatable

As per Section 4.6 of [RFC7252], the base value (and the value used when this option is not implemented) is 1152.

The active value of the Max-Message-Size Option is replaced each time the option is sent with a modified value. Its starting value is its base value.

5.3.2. Block-Wise-Transfer Capability Option

#	C	R	Applies to	Name	Format	Length	Base Value
4			CSM	Block-Wise-Transfer	empty	0	(none)

C=Critical, R=Repeatable

A sender can use the elective Block-Wise-Transfer Option to indicate that it supports the block-wise transfer protocol [RFC7959].

If the option is not given, the peer has no information about whether block-wise transfers are supported by the sender or not. An implementation wishing to offer block-wise transfers to its peer therefore needs to indicate the Block-Wise-Transfer Option.

If a Max-Message-Size Option is indicated with a value that is greater than 1152 (in the same or a different CSM message), the Block-Wise-Transfer Option also indicates support for BERT (see Section 6). Subsequently, if the Max-Message-Size Option is indicated with a value equal to or less than 1152, BERT support is no longer indicated. (Note that indication of BERT support obliges neither peer to actually choose to make use of BERT.)

Implementation note: When indicating a value of the Max-Message-Size option with an intention to enable BERT, the indicating implementation may want to choose a BERT size message it wants to encourage and add a delta for the header and any options that also need to be included in the message. Section 4.6 of [RFC7252] adds 128 bytes to a maximum block size of 1024 to arrive at a default message size of 1152. A BERT-enabled implementation may want to indicate a BERT block size of 2048 or a higher multiple of 1024, and at the same time be more generous for the size of header and options added (say, 256 or 512). Adding 1024 or more however to the base BERT block size may encourage the peer implementation to vary the BERT block size based on the size of the options included, which can be harder to establish interoperability for.

5.4. Ping and Pong Messages

In CoAP over reliable transports, Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient. This provides a basic keep-alive function. In contrast, Ping and Pong messages are a bidirectional exchange.

Upon receipt of a Ping message, the receiver MUST return a Pong message with an identical token in response. Unless the Ping carries an option with delaying semantics such as the Custody Option, it SHOULD respond as soon as practical. As with all Signaling messages, the recipient of a Ping or Pong message MUST ignore elective options it does not understand.

Ping and Pong messages are indicated by the 7.02 code (Ping) and the 7.03 code (Pong).

Note that, as with similar mechanisms defined in [RFC6455] and [RFC7540], the present specification does not define any specific maximum time that the sender of a Ping message has to allow waiting for a Pong reply. Any limitations on the patience for this reply are a matter of the application making use of these messages, as is any approach to recover from a failure to respond in time.

5.4.1. Custody Option

#	C	R	Applies to	Name	Format	Length	Base Value
2			Ping, Pong	Custody	empty	0	(none)

C=Critical, R=Repeatable

When responding to a Ping message, the receiver can include an elective Custody Option in the Pong message. This option indicates that the application has processed all the request/response messages received prior to the Ping message on the current connection. (Note that there is no definition of specific application semantics for "processed", but there is an expectation that the receiver of a Pong Message with a Custody Option should be able to free buffers based on this indication.)

A sender can also include an elective Custody Option in a Ping message to explicitly request the inclusion of an elective Custody Option in the corresponding Pong message. In that case, the receiver

SHOULD delay its Pong message until it finishes processing all the request/response messages received prior to the Ping message on the current connection.

5.5. Release Messages

A Release message indicates that the sender does not want to continue maintaining the transport connection and opts for an orderly shutdown, but wants to leave it to the peer to actually start closing the connection. The details are in the options. A diagnostic payload (see Section 5.5.2 of [RFC7252]) MAY be included.

A peer will normally respond to a Release message by closing the transport connection. (In case that does not happen, the sender of the release may want to implement a timeout mechanism if getting rid of the connection is actually important to it.)

Messages may be in flight or responses outstanding when the sender decides to send a Release message (which is one reason the sender had decided to wait with closing the connection). The peer responding to the Release message SHOULD delay the closing of the connection until it has responded to all requests received by it before the Release message. It also MAY wait for the responses to its own requests.

It is NOT RECOMMENDED for the sender of a Release message to continue sending requests on the connection it already indicated to be released: the peer might close the connection at any time and miss those requests. There is no obligation for the peer to check for this condition, though.

Release messages are indicated by the 7.04 code (Release).

Release messages can indicate one or more reasons using elective options. The following options are defined:

#	C	R	Applies to	Name	Format	Length	Base Value
2		x	Release	Alternative-Address	string	1-255	(none)

C=Critical, R=Repeatable

The elective Alternative-Address Option requests the peer to instead open a connection of the same scheme as the present connection to the alternative transport address given. Its value is in the form

"authority" as defined in Section 3.2 of [RFC3986]. (Existing state related to the connection is not transferred from the present connection to the new connection.)

The Alternative-Address Option is a repeatable option as defined in Section 5.4.5 of [RFC7252]. When multiple occurrences of the option are included, the peer can choose any of the alternative transport addresses.

#	C	R	Applies to	Name	Format	Length	Base Value
4			Release	Hold-Off	uint	0-3	(none)

C=Critical, R=Repeatable

The elective Hold-Off Option indicates that the server is requesting that the peer not reconnect to it for the number of seconds given in the value.

5.6. Abort Messages

An Abort message indicates that the sender is unable to continue maintaining the transport connection and cannot even wait for an orderly release. The sender shuts down the connection immediately after the abort (and may or may not wait for a Release or Abort message or connection shutdown in the inverse direction). A diagnostic payload (see Section 5.5.2 of [RFC7252]) SHOULD be included in the Abort message. Messages may be in flight or responses outstanding when the sender decides to send an Abort message. The general expectation is that these will NOT be processed.

Abort messages are indicated by the 7.05 code (Abort).

Abort messages can indicate one or more reasons using elective options. The following option is defined:

#	C	R	Applies to	Name	Format	Length	Base Value
2			Abort	Bad-CSM-Option	uint	0-2	(none)

C=Critical, R=Repeatable

The elective Bad-CSM-Option Option indicates that the sender is unable to process the CSM option identified by its option number, e.g. when it is critical and the option number is unknown by the sender, or when there is parameter problem with the value of an elective option. More detailed information SHOULD be included as a diagnostic payload.

For CoAP over UDP, messages which contain syntax violations are processed as message format errors. As described in Sections 4.2 and 4.3 of [RFC7252], such messages are rejected by sending a matching Reset message and otherwise ignoring the message.

For CoAP over reliable transports, the recipient rejects such messages by sending an Abort message and otherwise ignoring (not processing) the message. No specific option has been defined for the Abort message in this case, as the details are best left to a diagnostic payload.

5.7. Signaling examples

An encoded example of a Ping message with a non-empty token is shown in Figure 11.

```

0          1          2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           0x01           |           0xe2           |           0x42           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

Len    =    0 -----> 0x01
TKL    =    1 ____/
Code   = 7.02 Ping --> 0xe2
Token  =                      0x42

```

Figure 11: Ping Message Example

An encoded example of the corresponding Pong message is shown in Figure 12.


```

      0               1               2
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           0x01           |           0xe3           |           0x42           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

Len   =   0 -----> 0x01
TKL   =   1 ____/
Code  = 7.03 Pong --> 0xe3
Token =                   0x42

```

Figure 12: Pong Message Example

6. Block-wise Transfer and Reliable Transports

The message size restrictions defined in Section 4.6 of CoAP [RFC7252] to avoid IP fragmentation are not necessary when CoAP is used over a reliable transport. While this suggests that the Block-wise transfer protocol [RFC7959] is also no longer needed, it remains applicable for a number of cases:

- o large messages, such as firmware downloads, may cause undesired head-of-line blocking when a single transport connection is used
- o a UDP-to-TCP gateway may simply not have the context to convert a message with a Block Option into the equivalent exchange without any use of a Block Option (it would need to convert the entire blockwise exchange from start to end into a single exchange)

The 'Block-wise Extension for Reliable Transport (BERT)' extends the Block protocol to enable the use of larger messages over a reliable transport.

The use of this new extension is signaled by sending Block1 or Block2 Options with SZX == 7 (a "BERT option"). SZX == 7 is a reserved value in [RFC7959].

In control usage, a BERT option is interpreted in the same way as the equivalent Option with SZX == 6, except that it also indicates the capability to process BERT blocks. As with the basic Block protocol, the recipient of a CoAP request with a BERT option in control usage is allowed to respond with a different SZX value, e.g. to send a non-BERT block instead.

In descriptive usage, a BERT Option is interpreted in the same way as the equivalent Option with SZX == 6, except that the payload is also allowed to contain multiple blocks. For non-final BERT blocks, the payload is always a multiple of 1024 bytes. For final BERT blocks,

the payload is a multiple (possibly 0) of 1024 bytes plus a partial block of less than 1024 bytes.

The recipient of a non-final BERT block ($M=1$) conceptually partitions the payload into a sequence of 1024-byte blocks and acts exactly as if it had received this sequence in conjunction with block numbers starting at, and sequentially increasing from, the block number given in the Block Option. In other words, the entire BERT block is positioned at the byte position that results from multiplying the block number with 1024. The position of further blocks to be transferred is indicated by incrementing the block number by the number of elements in this sequence (i.e., the size of the payload divided by 1024 bytes).

As with $SZX == 6$, the recipient of a final BERT block ($M=0$) simply appends the payload at the byte position that is indicated by the block number multiplied with 1024.

The following examples illustrate BERT options. A value of $SZX == 7$ is labeled as "BERT" or as "BERT(nnn)" to indicate a payload of size nnn.

In all these examples, a Block Option is decomposed to indicate the kind of Block Option (1 or 2) followed by a colon, the block number (NUM), more bit (M), and block size ($2^{*(SZX+4)}$) separated by slashes. E.g., a Block2 Option value of 33 would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

6.1. Example: GET with BERT Blocks

Figure 13 shows a GET request with a response that is split into three BERT blocks. The first response contains 3072 bytes of payload; the second, 5120; and the third, 4711. Note how the block number increments to move the position inside the response body forward.

CoAP Client	CoAP Server
GET, /status	----->
<----- 2.05 Content, 2:0/1/BERT(3072)	
GET, /status, 2:3/0/BERT	----->
<----- 2.05 Content, 2:3/1/BERT(5120)	
GET, /status, 2:8/0/BERT	----->
<----- 2.05 Content, 2:8/0/BERT(4711)	

Figure 13: GET with BERT blocks

6.2. Example: PUT with BERT Blocks

Figure 14 demonstrates a PUT exchange with BERT blocks.

CoAP Client	CoAP Server
PUT, /options, 1:0/1/BERT(8192)	----->
<----- 2.31 Continue, 1:0/1/BERT	
PUT, /options, 1:8/1/BERT(16384)	----->
<----- 2.31 Continue, 1:8/1/BERT	
PUT, /options, 1:24/0/BERT(5683)	----->
<----- 2.04 Changed, 1:24/0/BERT	

Figure 14: PUT with BERT blocks

7. Observing Resources over Reliable Transports

This section describes how the procedures defined in [RFC7641] for observing resources over CoAP are applied (and modified, as needed) for reliable transports. In this section, "client" and "server" refer to the CoAP client and CoAP server.

7.1. Notifications and Reordering

When using the Observe Option with CoAP over UDP, notifications from the server set the option value to an increasing sequence number for reordering detection on the client since messages can arrive in a different order than they were sent. This sequence number is not required for CoAP over reliable transports since the TCP protocol ensures reliable and ordered delivery of messages. The value of the Observe Option in 2.xx notifications MAY be empty on transmission and MUST be ignored on reception.

Implementation note: This means that a proxy from a reordering transport to a reliable (in-order) transport (such as a UDP-to-TCP proxy) needs to process the Observe Option in notifications according to the rules in Section 3.4 of [RFC7641].

7.2. Transmission and Acknowledgements

For CoAP over UDP, server notifications to the client can be confirmable or non-confirmable. A confirmable message requires the client to either respond with an acknowledgement message or a reset message. An acknowledgement message indicates that the client is alive and wishes to receive further notifications. A reset message indicates that the client does not recognize the token which causes the server to remove the associated entry from the list of observers.

Since TCP eliminates the need for the message layer to support reliability, CoAP over reliable transports does not support confirmable or non-confirmable message types. All notifications are delivered reliably to the client with positive acknowledgement of receipt occurring at the TCP level. If the client does not recognize the token in a notification, it MAY immediately abort the connection (see Section 5.6).

7.3. Freshness

For CoAP over UDP, if a client does not receive a notification for some time, it MAY send a new GET request with the same token as the original request to re-register its interest in a resource and verify that the server is still responsive. For CoAP over reliable transports, it is more efficient to check the health of the connection (and all its active observations) by sending a single CoAP Ping Signaling message (Section 5.4) rather than individual requests to confirm each active observation. (Note that such a Ping/Pong only confirms a single hop: there is no obligation, and no expectation, of a proxy to react to a Ping by checking all its onward observations or all the connections, if any, underlying them. A proxy MAY maintain its own schedule for confirming the onward observations it relies on;

it is however generally inadvisable for a proxy to generate a large number of outgoing checks based on a single incoming check.)

7.4. Cancellation

For CoAP over UDP, a client that is no longer interested in receiving notifications can "forget" the observation and respond to the next notification from the server with a reset message to cancel the observation.

For CoAP over reliable transports, a client MUST explicitly deregister by issuing a GET request that has the Token field set to the token of the observation to be cancelled and includes an Observe Option with the value set to 1 (deregister).

If the client observes one or more resources over a reliable transport, then the CoAP server (or intermediary in the role of the CoAP server) MUST remove all entries associated with the client endpoint from the lists of observers when the connection is either closed or times out.

8. CoAP over Reliable Transport URIs

CoAP over UDP [RFC7252] defines the "coap" and "coaps" URI schemes. This document introduces four additional URI schemes for identifying CoAP resources and providing a means of locating the resource:

- o the "coap+tcp" URI scheme for CoAP over TCP
- o the "coaps+tcp" URI scheme for CoAP over TCP secured by TLS
- o the "coap+ws" URI scheme for CoAP over WebSockets
- o the "coaps+ws" URI scheme for CoAP over WebSockets secured by TLS

Resources made available via these schemes have no shared identity even if their resource identifiers indicate the same authority (the same host listening to the same TCP port). They are hosted in distinct namespaces because each URI scheme implies a distinct origin server.

The syntax for the URI schemes in this section are specified using Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", and "query" are adopted from [RFC3986].

Section 8 (Multicast CoAP) in [RFC7252] is not applicable to these schemes.

As with the "coap" and "coaps" schemes defined in [RFC7252], all URI schemes defined in this section also support the path prefix `"/.well-known/"` defined by [RFC5785] for "well-known locations" in the namespace of a host. This enables discovery as per Section 7 of [RFC7252].

8.1. coap+tcp URI scheme

The "coap+tcp" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over TCP.

```
coap-tcp-URI = "coap+tcp:" "/" host [ ":" port ]
               path-abempty [ "?" query ]
```

The syntax defined in Section 6.1 of [RFC7252] applies to this URI scheme with the following changes:

- o The port subcomponent indicates the TCP port at which the CoAP Connection Acceptor is located. (If it is empty or not given, then the default port 5683 is assumed, as with UDP.)

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.2. coaps+tcp URI scheme

The "coaps+tcp" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over TCP secured with TLS.

```
coaps-tcp-URI = "coaps+tcp:" "/" host [ ":" port ]
                path-abempty [ "?" query ]
```

The syntax defined in Section 6.2 of [RFC7252] applies to this URI scheme, with the following changes:

- o The port subcomponent indicates the TCP port at which the TLS server for the CoAP Connection Acceptor is located. If it is empty or not given, then the default port 5684 is assumed.
- o If a TLS server does not support the Application-Layer Protocol Negotiation Extension (ALPN) [RFC7301] or wishes to accommodate TLS clients that do not support ALPN, it MAY offer a coaps+tcp endpoint on TCP port 5684. This endpoint MAY also be ALPN

enabled. A TLS server MAY offer coaps+tcp endpoints on ports other than TCP port 5684, which MUST be ALPN enabled.

- o For TCP ports other than port 5684, the TLS client MUST use the ALPN extension to advertise the "coap" protocol identifier (see Section 11.7) in the list of protocols in its ClientHello. If the TCP server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the TLS server either does not negotiate the ALPN extension or returns a no_application_protocol alert, the TLS client MUST close the connection.
- o For TCP port 5684, a TLS client MAY use the ALPN extension to advertise the "coap" protocol identifier in the list of protocols in its ClientHello. If the TLS server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the TLS server returns a no_application_protocol alert, then the TLS client MUST close the connection. If the TLS server does not negotiate the ALPN extension, then coaps+tcp is implicitly selected.
- o For TCP port 5684, if the TLS client does not use the ALPN extension to negotiate the protocol, then coaps+tcp is implicitly selected.

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.3. coap+ws URI scheme

The "coap+ws" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over WebSockets.

```
coap-ws-URI = "coap+ws:" "/" host [ ":" port ]
              path-abempty [ "?" query ]
```

The port subcomponent is OPTIONAL. The default is port 80.

The WebSocket endpoint is identified by a "ws" URI that is composed of the authority part of the "coap+ws" URI and the well-known path `"/.well-known/coap"` [RFC5785] [I-D.bormann-hybi-ws-wk]. The path and query parts of a "coap+ws" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP:

```

coap+ws://example.org/sensors/temperature?u=Cel
      \_____/ \_____/ \_____/
      \_/      \_/      \_/
ws://example.org/.well-known/coap  Uri-Path: "sensors"
                                   Uri-Path: "temperature"
                                   Uri-Query: "u=Cel"

```

Figure 15: The "coap+ws" URI Scheme

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.4. coaps+ws URI scheme

The "coaps+ws" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over WebSockets secured by TLS.

```

coaps-ws-URI = "coaps+ws:" "/" host [ ":" port ]
path-abempty [ "?" query ]

```

The port subcomponent is OPTIONAL. The default is port 443.

The WebSocket endpoint is identified by a "wss" URI that is composed of the authority part of the "coaps+ws" URI and the well-known path `"/.well-known/coap"` [RFC5785] [I-D.bormann-hybi-ws-wk]. The path and query parts of a "coaps+ws" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP.

```

coaps+ws://example.org/sensors/temperature?u=Cel
      \_____/ \_____/ \_____/
      \_/      \_/      \_/
wss://example.org/.well-known/coap  Uri-Path: "sensors"
                                   Uri-Path: "temperature"
                                   Uri-Query: "u=Cel"

```

Figure 16: The "coaps+ws" URI Scheme

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.5. Uri-Host and Uri-Port Options

CoAP over reliable transports maintains the property from Section 5.10.1 of [RFC7252]:

The default values for the Uri-Host and Uri-Port Options are sufficient for requests to most servers.

Unless otherwise noted, the default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. The default value of the Uri-Port Option is the destination TCP port.

For CoAP over TLS, these default values are the same unless Server Name Indication (SNI) [RFC6066] is negotiated. In this case, the default value of the Uri-Host Option in requests from the TLS client to the TLS server is the SNI host.

For CoAP over WebSockets, the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server is indicated by the Host header field from the WebSocket handshake.

8.6. Decomposing URIs into Options

The steps are the same as specified in Section 6.4 of [RFC7252] with minor changes.

This step from [RFC7252]:

3. If `|url|` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.

is updated to:

3. If `|url|` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap+tcp", "coaps+tcp", "coap+ws", or "coaps+ws", then fail this algorithm.

This step from [RFC7252]:

7. If `|port|` does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be `|port|`.

is updated to:

7. If `|port|` does not equal the request's destination TCP port, include a Uri-Port Option and let that option's value be `|port|`.

8.7. Composing URIs from Options

The steps are the same as specified in Section 6.5 of [RFC7252] with minor changes.

This step from [RFC7252]:

1. If the request is secured using DTLS, let `|url|` be the string "coaps://". Otherwise, let `|url|` be the string "coap://".

is updated to:

1. For CoAP over TCP, if the request is secured using TLS, let `|url|` be the string "coaps+tcp://". Otherwise, let `|url|` be the string "coap+tcp://". For CoAP over WebSockets, if the request is secured using TLS, let `|url|` be the string "coaps+ws://". Otherwise, let `|url|` be the string "coap+ws://".

This step from [RFC7252]:

4. If the request includes a Uri-Port Option, let `|port|` be that option's value. Otherwise, let `|port|` be the request's destination UDP port.

is updated to:

4. If the request includes a Uri-Port Option, let `|port|` be that option's value. Otherwise, let `|port|` be the request's destination TCP port.

9. Securing CoAP

Security Challenges for the Internet of Things [SecurityChallenges] recommends:

... it is essential that IoT protocol suites specify a mandatory to implement but optional to use security solution. This will ensure security is available in all implementations, but configurable to use when not necessary (e.g., in closed environment). ... even if those features stretch the capabilities of such devices.

A security solution **MUST** be implemented to protect CoAP over reliable transports and **MUST** be enabled by default. This document defines the TLS binding, but alternative solutions at different layers in the protocol stack **MAY** be used to protect CoAP over reliable transports when appropriate. Note that there is ongoing work to support a data

object-based security model for CoAP that is independent of transport (see [I-D.ietf-core-object-security]).

9.1. TLS binding for CoAP over TCP

The TLS usage guidance in [RFC7925] applies, including the guidance about cipher suites in that document that are derived from the mandatory-to-implement (MTI) cipher suites defined in [RFC7525].

This guidance assumes implementation in a constrained device or for communication with a constrained device. CoAP over TCP/TLS has, however, a wider applicability. It may, for example, be implemented on a gateway or on a device that is less constrained (such as a smart phone or a tablet), for communication with a peer that is likewise less constrained, or within a backend environment that only communicates with constrained devices via proxies. As an exception to the previous paragraph, in this case, the recommendations in [RFC7525] are more appropriate.

Since the guidance offered in [RFC7925] and [RFC7525] differs in terms of algorithms and credential types, it is assumed that a CoAP over TCP/TLS implementation that needs to support both cases implements the recommendations offered by both specifications.

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials, access control lists, and authorization servers. At the end of the provisioning phase, the device will be in one of four security modes:

NoSec: TLS is disabled.

PreSharedKey: TLS is enabled. The guidance in Section 4.2 of [RFC7925] applies.

RawPublicKey: TLS is enabled. The guidance in Section 4.3 of [RFC7925] applies.

Certificate: TLS is enabled. The guidance in Section 4.4 of [RFC7925] applies.

The "NoSec" mode is optional-to-implement. The system simply sends the packets over normal TCP which is indicated by the "coap+tcp" scheme and the TCP CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes.

"PreSharedKey", "RawPublicKey", or "Certificate" is mandatory-to-implement for the TLS binding depending on the credential type used

with the device. These security modes are achieved using TLS and are indicated by the "coaps+tcp" scheme and TLS-secured CoAP default port.

9.2. TLS usage for CoAP over WebSockets

A CoAP client requesting a resource identified by a "coaps+ws" URI negotiates a secure WebSocket connection to a WebSocket server endpoint with a "wss" URI. This is described in Section 8.4.

The client MUST perform a TLS handshake after opening the connection to the server. The guidance in Section 4.1 of [RFC6455] applies. When a CoAP server exposes resources identified by a "coaps+ws" URI, the guidance in Section 4.4 of [RFC7925] applies towards mandatory-to-implement TLS functionality for certificates. For the server-side requirements in accepting incoming connections over a HTTPS (HTTP-over-TLS) port, the guidance in Section 4.2 of [RFC6455] applies.

Note that this formally inherits the mandatory-to-implement cipher suites defined in [RFC5246]. However, usually modern browsers implement more recent cipher suites that then are automatically picked up via the JavaScript WebSocket API. WebSocket Servers that provide Secure CoAP over WebSockets for the browser use case will need to follow the browser preferences and MUST follow [RFC7525].

10. Security Considerations

The security considerations of [RFC7252] apply. For CoAP over WebSockets and CoAP over TLS-secured WebSockets, the security considerations of [RFC6455] also apply.

10.1. Signaling Messages

The guidance given by an Alternative-Address Option cannot be followed blindly. In particular, a peer MUST NOT assume that a successful connection to the Alternative-Address inherits all the security properties of the current connection.

11. IANA Considerations

11.1. Signaling Codes

IANA is requested to create a third sub-registry for values of the Code field in the CoAP header (Section 12.1 of [RFC7252]). The name of this sub-registry is "CoAP Signaling Codes".

Each entry in the sub-registry must include the Signaling Code in the range 7.00-7.31, its name, and a reference to its documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
7.01	CSM	[RFCthis]
7.02	Ping	[RFCthis]
7.03	Pong	[RFCthis]
7.04	Release	[RFCthis]
7.05	Abort	[RFCthis]

Table 1: CoAP Signal Codes

All other Signaling Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC8126].

11.2. CoAP Signaling Option Numbers Registry

IANA is requested to create a sub-registry for Options Numbers used in CoAP signaling options within the "CoRE Parameters" registry. The name of this sub-registry is "CoAP Signaling Option Numbers".

Each entry in the sub-registry must include one or more of the codes in the Signaling Codes subregistry (Section 11.1), the option number, the name of the option, and a reference to the option's documentation.

Initial entries in this sub-registry are as follows:

Applies to	Number	Name	Reference
7.01	2	Max-Message-Size	[RFCthis]
7.01	4	Block-Wise-Transfer	[RFCthis]
7.02, 7.03	2	Custody	[RFCthis]
7.04	2	Alternative-Address	[RFCthis]
7.04	4	Hold-Off	[RFCthis]
7.05	2	Bad-CSM-Option	[RFCthis]

Table 2: CoAP Signal Option Codes

The IANA policy for future additions to this sub-registry is based on number ranges for the option numbers, analogous to the policy defined in Section 12.2 of [RFC7252]. (The policy is analogous rather than identical because the structure of the subregistry includes an additional column; however, the value of this column has no influence on the policy.)

The documentation for a Signaling Option Number should specify the semantics of an option with that number, including the following properties:

- o Whether the option is critical or elective, as determined by the Option Number.
- o Whether the option is repeatable.
- o The format and length of the option's value.
- o The base value for the option, if any.

11.3. Service Name and Port Number Registration

IANA is requested to assign the port number 5683 and the service name "coap+tcp", in accordance with [RFC6335].

Service Name.
coap+tcp

Transport Protocol.
tcp

Assignee.

IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

Constrained Application Protocol (CoAP)

Reference.

[RFCthis]

Port Number.

5683

11.4. Secure Service Name and Port Number Registration

IANA is requested to assign the port number 5684 and the service name "coaps+tcp", in accordance with [RFC6335]. The port number is requested to address the exceptional case of TLS implementations that do not support the "Application-Layer Protocol Negotiation Extension" [RFC7301].

Service Name.

coaps+tcp

Transport Protocol.

tcp

Assignee.

IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

Constrained Application Protocol (CoAP)

Reference.

[RFC7301], [RFCthis]

Port Number.

5684

11.5. URI Scheme Registration

URI schemes are registered within the "Uniform Resource Identifier (URI) Schemes" registry maintained at [IANA.uri-schemes].

11.5.1. coap+tcp

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coap+tcp". This registration request complies with [RFC7595].

Scheme name:
coap+tcp

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using TCP.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.1 in [RFCthis]

11.5.2. coaps+tcp

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coaps+tcp". This registration request complies with [RFC7595].

Scheme name:
coaps+tcp

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using TLS.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.2 in [RFCthis]

11.5.3. coap+ws

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coap+ws". This registration request complies with [RFC7595].

Scheme name:
coap+ws

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.3 in [RFCthis]

11.5.4. coaps+ws

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coaps+ws". This registration request complies with [RFC7595].

Scheme name:
coaps+ws

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

References:
Section 8.4 in [RFCthis]

11.6. Well-Known URI Suffix Registration

IANA is requested to register the 'coap' well-known URI in the "Well-Known URIs" registry. This registration request complies with [RFC5785]:

URI Suffix.
coap

Change controller.
IETF

Specification document(s).
[RFCthis]

Related information.
None.

11.7. ALPN Protocol Identifier

IANA is requested to assign the following value in the registry "Application Layer Protocol Negotiation (ALPN) Protocol IDs" created by [RFC7301]. The "coap" string identifies CoAP when used over TLS.

Protocol.
CoAP

Identification Sequence.
0x63 0x6f 0x61 0x70 ("coap")

Reference.
[RFCthis]

11.8. WebSocket Subprotocol Registration

IANA is requested to register the WebSocket CoAP subprotocol under the "WebSocket Subprotocol Name Registry":

Subprotocol Identifier.
coap

Subprotocol Common Name.

Constrained Application Protocol (CoAP)

Subprotocol Definition.
[RFCthis]

11.9. CoAP Option Numbers Registry

IANA is requested to add [RFCthis] to the references for the following entries registered by [RFC7959] in the "CoAP Option Numbers" sub-registry defined by [RFC7252]:

Number	Name	Reference
23	Block2	RFC 7959, [RFCthis]
27	Block1	RFC 7959, [RFCthis]

Table 3: CoAP Option Numbers

12. References

12.1. Normative References

- [I-D.bormann-hybi-ws-wk]
Bormann, C., "Well-known URIs for the WebSocket Protocol", draft-bormann-hybi-ws-wk-00 (work in progress), May 2017.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.

- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/info/rfc7301>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<https://www.rfc-editor.org/info/rfc7595>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.

- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.

12.2. Informative References

- [BK2015] Byrne, C. and J. Kleberg, "Advisory Guidelines for UDP Deployment", Proceedings draft-byrne-opsec-udp-advisory-00 (expired), 2015.
- [EK2016] Edeline, K., Kuehlewind, M., Trammell, B., Aben, E., and B. Donnet, "Using UDP for Internet Transport Evolution", Proceedings arXiv preprint 1612.07816, 2016.
- [HomeGateway] Eggert, L., "An experimental study of home gateway characteristics", Proceedings of the 10th annual conference on Internet measurement , 2010.
- [I-D.gomez-lwig-tcp-constrained-node-networks] Gomez, C., Crowcroft, J., and M. Scharf, "TCP over Constrained-Node Networks", draft-gomez-lwig-tcp-constrained-node-networks-03 (work in progress), June 2017.
- [I-D.ietf-core-cocoa] Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-02 (work in progress), October 2017.
- [I-D.ietf-core-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-07 (work in progress), November 2017.
- [IANA.uri-schemes] IANA, "Uniform Resource Identifier (URI) Schemes", <<http://www.iana.org/assignments/uri-schemes>>.

- [LWM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification Version 1.0", February 2017, <http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-TS-LightweightM2M-V1_0-20170208-A.pdf>.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [SecurityChallenges] Polk, T. and S. Turner, "Security Challenges for the Internet of Things", Interconnecting Smart Objects with the Internet / IAB Workshop , February 2011, <<http://www.iab.org/wp-content/IAB-uploads/2011/03/Turner.pdf>>.
- [SW2016] Swett, I., "QUIC Deployment Experience @Google", Proceedings <https://www.ietf.org/proceedings/96/slides/slides-96-quic-3.pdf>, 2016.

Appendix A. CoAP over WebSocket Examples

This section gives examples for the first two configurations discussed in Section 4.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be as follows. Figure 17 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap+ws://example.org/sensors/temperature?u=Cel>, for example, from a resource representation that it retrieved previously.
2. It establishes a WebSocket connection to the endpoint URI composed of the authority "example.org" and the well-known path "/.well-known/coap", <ws://example.org/.well-known/coap>.
3. CSM messages (Section 5.3) are exchanged (not shown for lack of space).
4. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.
5. It waits for the server to return a response.
6. The CoAP client uses the connection for further requests, or the connection is closed.

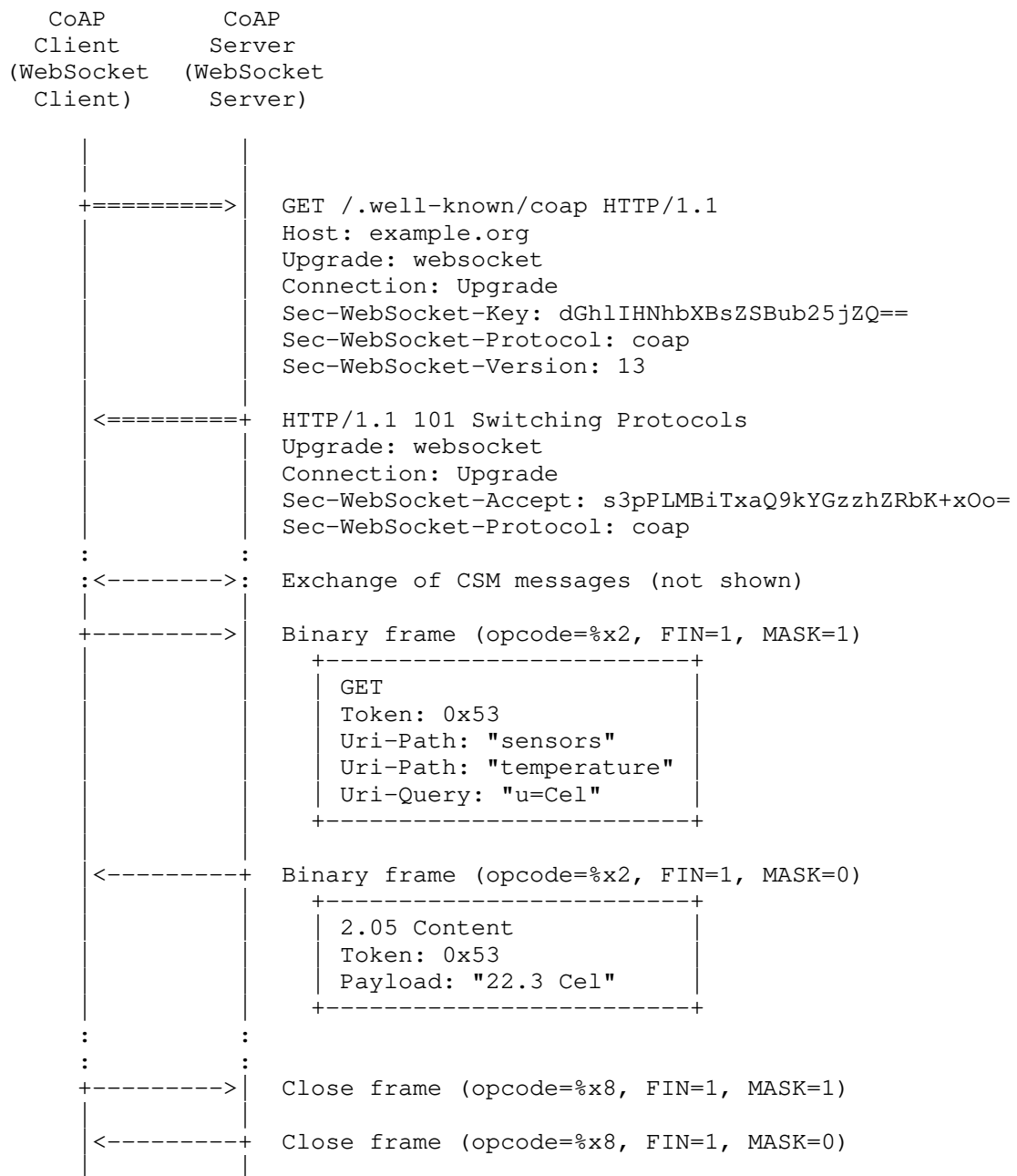


Figure 17: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 18 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource "coap://[2001:db8::1]/". The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response over the WebSocket connection to the client.

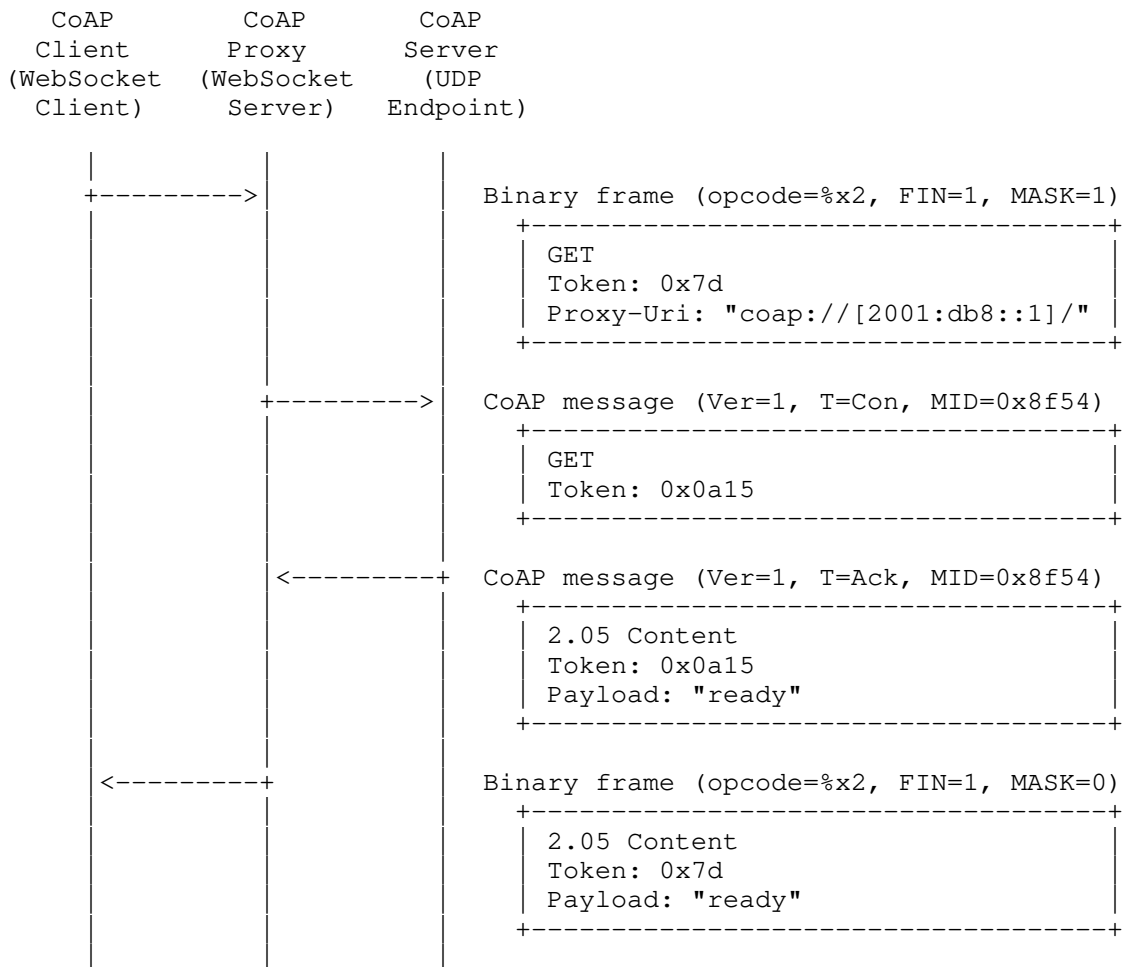


Figure 18: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSocket-enabled CoAP proxy

Appendix B. Change Log

The RFC Editor is requested to remove this section at publication.

B.1. Since draft-ietf-core-coap-tcp-tls-02

Merged draft-savolainen-core-coap-websockets-07 Merged draft-bormann-core-block-bert-01 Merged draft-bormann-core-coap-sig-02

B.2. Since draft-ietf-core-coap-tcp-tls-03

Editorial updates

Added mandatory exchange of Capabilities and Settings messages after connecting

Added support for coaps+tcp port 5684 and more details on Application-Layer Protocol Negotiation (ALPN)

Added guidance on CoAP Signaling Ping-Pong versus WebSocket Ping-Pong

Updated references and requirements for TLS security considerations

B.3. Since draft-ietf-core-coap-tcp-tls-04

Updated references

Added Appendix: Updates to RFC7641 Observing Resources in the Constrained Application Protocol (CoAP)

Updated Capability and Settings Message (CSM) exchange in the Opening Handshake to allow initiator to send messages before receiving acceptor CSM

B.4. Since draft-ietf-core-coap-tcp-tls-05

Addressed feedback from Working Group Last Call

Added Securing CoAP section and informative reference to OSCOAP

Removed the Server-Name and Bad-Server-Name Options

Clarified the Capability and Settings Message (CSM) exchange

Updated Pong response requirements

Added Connection Initiator and Connection Acceptor terminology where appropriate

Updated LWM2M 1.0 informative reference

B.5. Since draft-ietf-core-coap-tcp-tls-06

Addressed feedback from second Working Group Last Call

B.6. Since draft-ietf-core-coap-tcp-tls-07

Addressed feedback from IETF Last Call

Addressed feedback from ARTART review

Addressed feedback from GENART review

Addressed feedback from TSVART review

Added fragment identifiers to URI schemes

Added "Updates RFC7959" for BERT

Added "Updates RFC6455" to extend well-known URI mechanism to ws and wss

Clarified well-known URI mechanism use for all URI schemes

Changed NoSec to optional-to-implement

Acknowledgements

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Esko Dijk, Christian Groves, Nadir Javed, Michael Koster, Matthias Kovatsch, Achim Kraus, David Navarro, Szymon Sasin, Goran Selander, Zach Shelby, Andrew Summers, Julien Vermillard, and Gengyu Wei for their feedback.

Last-call reviews from Yoshifumi Nishida, Mark Nottingham, and Meral Shirazipour as well as several IESG reviewers provided extensive comments; from the IESG, we would like to specifically call out Ben Campbell, Mirja Kuehlewind, Eric Rescorla, Adam Roach, and the responsible AD Alexey Melnikov.

Contributors

Matthias Kovatsch
Siemens AG
Otto-Hahn-Ring 6
Munich D-81739

Phone: +49-173-5288856
EMail: matthias.kovatsch@siemens.com

Teemu Savolainen
Nokia Technologies
Hatanpaan valtatie 30
Tampere FI-33100
Finland

Email: teemu.savolainen@nokia.com

Valik Solorzano Barboza
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: vsolorzanobarboza@zebra.com

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Simon Lemay
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: slemay@zebra.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
Great Britain

Email: Hannes.tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

Brian Raymor (editor)

Email: brianraymor@hotmail.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 19, 2016

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
Alcatel-Lucent
E. Dijk
Philips Research
March 18, 2016

Guidelines for HTTP-CoAP Mapping Implementations
draft-ietf-core-http-mapping-08

Abstract

This document provides reference information for implementing a proxy that performs translation between the HTTP protocol and the CoAP protocol, focusing on the reverse proxy case. It describes how a HTTP request is mapped to a CoAP request and how a CoAP response is mapped back to a HTTP response. Furthermore, it defines a template for URI mapping and provides a set of guidelines for HTTP to CoAP protocol translation and related proxy implementations.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 19, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Reverse HTTP-CoAP Proxy	5
4. Use Cases	6
5. URI Mapping	7
5.1. URI Terminology	8
5.2. Default Mapping	8
5.2.1. Optional Scheme Omission	9
5.2.2. Encoding Caveats	9
5.3. URI Mapping Template	10
5.3.1. Simple Form	10
5.3.2. Enhanced Form	12
5.4. Discovery	13
5.4.1. Examples	14
6. Media Type Mapping	16
6.1. Overview	16
6.2. 'application/coap-payload' Media Type	17
6.3. Loose Media Type Mapping	18
6.4. Media Type to Content Format Mapping Algorithm	19
6.5. Content Transcoding	20
6.5.1. General	20
6.5.2. CoRE Link Format	20
6.5.3. Diagnostic Messages	21
7. Response Code Mapping	21
8. Additional Mapping Guidelines	24
8.1. Caching and Congestion Control	24
8.2. Cache Refresh via Observe	24
8.3. Use of CoAP Blockwise Transfer	25
8.4. Security Translation	26
8.5. CoAP Multicast	26
8.6. Timeouts	27
8.7. Miscellaneous	27
9. IANA Considerations	27
9.1. New 'core.hc' Resource Type	27
9.2. New 'coap-payload' Internet Media Type	28
10. Security Considerations	29

10.1. Traffic Overflow	29
10.2. Handling Secured Exchanges	30
10.3. Proxy and CoAP Server Resource Exhaustion	31
10.4. URI Mapping	31
11. Acknowledgements	32
12. References	32
12.1. Normative References	32
12.2. Informative References	33
Appendix A. Change Log	34
Authors' Addresses	37

1. Introduction

CoAP [RFC7252] has been designed with the twofold aim to be an application protocol specialized for constrained environments and to be easily used in Representational State Transfer (REST) based architectures such as the Web. The latter goal has led to defining CoAP to easily interoperate with HTTP [RFC7230] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [RFC7252] describes the fundamentals of the CoAP-to-HTTP and the HTTP-to-CoAP cross-protocol mapping process. However, [RFC7252] focuses primarily on specifying the forward proxy scenario, and leaves many aspects of the reverse proxy scenario for future definition. Therefore, a primary goal of this informational document is to define a consistent set of guidelines that an HTTP-to-CoAP reverse proxy implementation MAY adhere to. The main reason for adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability between an HTTP endpoint and a CoAP endpoint independent of the reverse proxy that implements the cross-protocol mapping. (For example, a reverse proxy conforming to these guidelines made by vendor A can be easily replaced by a reverse proxy from vendor B that also conforms to the guidelines.)

This document is organized as follows:

- o Section 2 describes terminology to identify proxy types, mapping approaches and proxy deployments;
- o Section 3 introduces the reverse HTTP-CoAP proxy;
- o Section 4 lists use cases in which HTTP clients need to contact CoAP servers;
- o Section 5 introduces a default and advanced HTTP-to-CoAP URI mapping syntax;

- o Section 6 describes how to map HTTP media types to CoAP content formats and vice versa;
- o Section 7 describes how to map CoAP responses to HTTP responses;
- o Section 8 describes additional mapping guidelines related to caching, congestion, timeouts and CoAP blockwise [I-D.ietf-core-block] transfers;
- o Section 10 discusses possible security impact of HTTP-CoAP protocol mapping.

2. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

HC Proxy: a proxy performing a cross-protocol mapping, in the context of this document a HTTP-CoAP mapping. A Cross-Protocol Proxy can behave as a Forward Proxy, Reverse Proxy or Interception Proxy. In this document we focus on the Reverse Proxy case.

Forward Proxy (or Forward HC Proxy): a message forwarding agent that is selected by the client, usually via local configuration rules, to receive requests for some type(s) of absolute URI and to attempt to satisfy those requests via translation to the protocol indicated by the absolute URI. The user decides (is willing to) use the proxy as the forwarding/de-referencing agent for a predefined subset of the URI space. In [RFC7230] this is called a Proxy. [RFC7252] defines Forward-Proxy similarly.

Reverse Proxy(or Reverse HC Proxy): as in [RFC7230], a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server's protocol. A Reverse HC Proxy behaves as an origin (HTTP) server on its connection towards the (HTTP) client and as a (CoAP) client on its connection towards the (CoAP) origin server. The (HTTP) client uses the "origin-form" (Section 5.3.1 of [RFC7230]) as a request-target URI.

Interception Proxy (or Interception HC Proxy) [RFC3040]: a proxy that receives inbound traffic flows through the process of traffic redirection; transparent to the client.

Placement terms: a Server-Side proxy is placed in the same network domain as the server; conversely a Client-Side proxy is placed in the

same network domain as the client. In any other case, the proxy is said to be External.

Note that a Reverse Proxy appears to a client as an origin server while a Forward Proxy does not, so, when communicating with a Reverse Proxy a client may be unaware it is communicating with a proxy at all.

3. Reverse HTTP-CoAP Proxy

A reverse HC proxy is accessed by clients only supporting HTTP, and handles their HTTP requests by mapping these to CoAP requests, which are forwarded to CoAP servers; mapping back received CoAP responses to HTTP responses. This mechanism is transparent to the client, which may assume that it is communicating with the intended target HTTP server. In other words, the client accesses the proxy as an origin server using the "origin-form" (Section 5.3.1 of [RFC7230]) as a request target.

See Figure 1 for an example deployment scenario. Here a reverse HC proxy is placed server-side, at the boundary of the Constrained Network domain, to avoid sending any HTTP traffic into the Constrained Network and to avoid any (unsecured) CoAP multicast traffic outside the Constrained Network. A DNS server (not shown) is used by the HTTP Client to resolve the IP address of the reverse HC proxy and optionally also used by the reverse HC proxy to resolve IP addresses of CoAP servers.

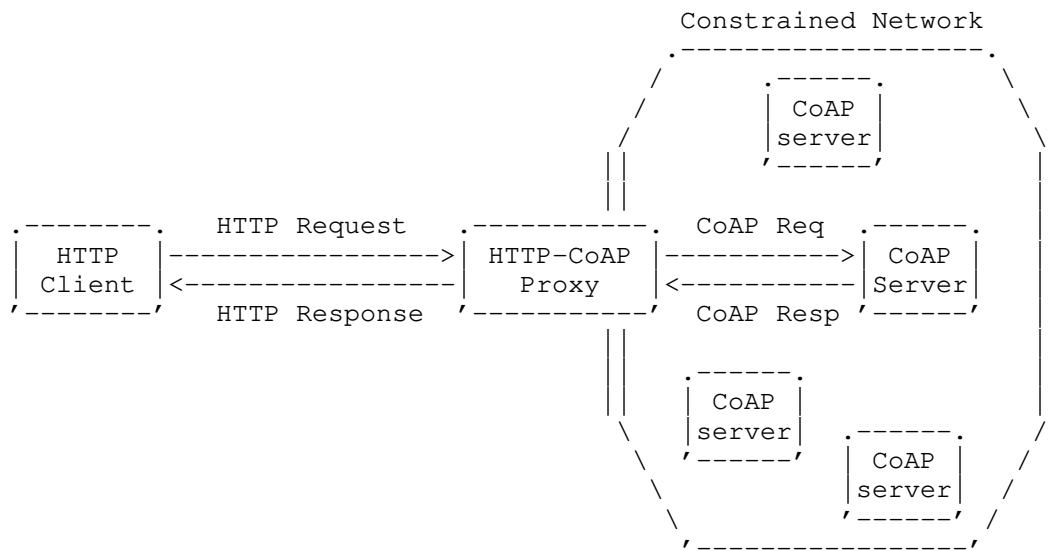


Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

Other placement options for the reverse HC proxy are client-side (not shown), which is in the same domain as the HTTP Client; or external, which is both outside the HTTP Client's domain and the CoAP servers' domain.

Normative requirements on the translation of HTTP requests to CoAP requests and of the CoAP responses back to HTTP responses are defined in Section 10.2 of [RFC7252]. However, [RFC7252] only considers the case of a Forward HC Proxy in which a client explicitly indicates it targets a request to a CoAP server. This document provides guidelines and more details for the implementation of a Reverse HC Proxy, which MAY be followed in addition to the normative requirements. Note that most of the guidelines also apply to an Intercepting HC Proxy.

4. Use Cases

To illustrate in which situations HTTP to CoAP protocol translation may be used, three use cases are described below.

1. Legacy building control application without CoAP: A building control application that uses HTTP but not CoAP, can check the status of CoAP sensors and/or actuators via a reverse HC proxy.

2. Making sensor data available to 3rd parties on the Web: For demonstration or public interest purposes, a reverse HC proxy may be configured to expose the contents of a CoAP sensor to the world via the web (HTTP and/or HTTPS). Some sensors might only handle secure 'coaps' requests, therefore the proxy is configured to translate any request to a 'coaps' secured request. The reverse HC proxy is furthermore configured to only pass through GET requests in order to protect the constrained network. In this way even unattended HTTP clients, such as web crawlers, may index sensor data as regular web pages.

3. Smartphone and home sensor: A smartphone can access directly a CoAP home sensor using an authenticated 'https' request, if its home router contains a reverse HC proxy. An HTML5 application on the smartphone can provide a friendly UI to the user using standard (HTTP) networking functions of HTML5.

A key point in the above use cases is the expected nature of the URI to be used by the HTTP client initiating the HTTP request to the reverse HC proxy. Specifically, in use case #1, there will be no "coap" or "coaps" related information embedded in the HTTP URI as it is a legacy HTTP client sending the request. So, the HTTP request will follow the processing steps described in all later sections of this document except for the one defined in section Section 5 (i.e., related to embedded "coap" or "coaps" URI processing). Use case #2 is also expected to be similar.

In contrast, in use case #3, it is expected that the HTTP client will specifically embed "coap" or "coaps" related information in the HTTP URI of the HTTP request to the reverse HC proxy. In this case, the HTTP request will follow the processing steps described in all later sections of this document including the one defined in section Section 5 (i.e., related to embedded "coap" or "coaps" URI processing).

5. URI Mapping

Though, in principle, a CoAP URI could be directly used by a HTTP client to de-reference a CoAP resource through a reverse HC proxy, the reality is that all major web browsers, networking libraries and command line tools do not allow making HTTP requests using URIs with a scheme "coap" or "coaps".

Thus, there is a need for web applications to embed or "pack" a CoAP URI into a HTTP URI so that it can be (non-destructively) transported from the HTTP client to the reverse HC proxy. The reverse HC proxy can then "unpack" the CoAP URI and finally de-reference it via a CoAP request to the target Server.

URI Mapping is the process through which the URI of a CoAP resource is transformed into an HTTP URI so that:

- o the requesting HTTP client can handle it;
- o the receiving reverse HC proxy can extract the intended CoAP URI unambiguously.

To this end, the remainder of this section will identify:

- o the default mechanism to map a CoAP URI into a HTTP URI;
- o the URI template format to express a class of CoAP-HTTP URI mapping functions;
- o the discovery mechanism based on CoRE Link Format [RFC6690] through which clients of a reverse HC proxy can dynamically discover information about the supported URI Mapping Template(s), as well as the URI where the reverse HC proxy function is anchored.

5.1. URI Terminology

In the remainder of this section, the following terms will be used with a distinctive meaning:

HC Proxy URI:

URI which refers to the reverse HC proxy function. It conforms to syntax defined in Section 2.7 of [RFC7230].

Target CoAP URI:

URI which refers to the (final) CoAP resource that has to be de-referenced. It conforms to syntax defined in Section 6 of [RFC7252]. Specifically, its scheme is either "coap" or "coaps".

Hosting HTTP URI:

URI that conforms to syntax in Section 2.7 of [RFC7230]. Its authority component refers to a reverse HC proxy, whereas path (and query) component(s) embed the information used by a reverse HC proxy to extract the Target CoAP URI.

5.2. Default Mapping

The default mapping is for the Target CoAP URI to be appended as-is to the HC Proxy URI, to form the Hosting HTTP URI. This is the URI that will then be sent by the HTTP client in the HTTP request to the reverse HC proxy.

For example: given a HC Proxy URI `http://p.example.com/hc` and a Target CoAP URI `coap://s.example.com/light`, the resulting Hosting HTTP URI would be `http://p.example.com/hc/coap://s.example.com/light`.

Provided a correct Target CoAP URI, the Hosting HTTP URI resulting from the default mapping is always syntactically correct. Furthermore, the Target CoAP URI can always be extracted unambiguously from the Hosting HTTP URI. Also, it is worth noting that, using the default mapping, a query component in the target CoAP resource URI is naturally encoded into the query component of the Hosting URI, e.g.: `coap://s.example.com/light?dim=5` becomes `http://p.example.com/hc/coap://s.example.com/light?dim=5`.

There is no default for the HC Proxy URI. Therefore, it is either known in advance, e.g. as a configuration preset, or dynamically discovered using the mechanism described in Section 5.4.

The default URI mapping function is RECOMMENDED to be implemented and activated by default in a reverse HC proxy, unless there are valid reasons, e.g. application specific, to use a different mapping function.

5.2.1. Optional Scheme Omission

When found in a Hosting HTTP URI, the scheme (i.e., "coap" or "coaps"), the scheme component delimiter (":"), and the double slash ("//") preceding the authority MAY be omitted. In such case, a local default - not defined by this document - applies.

So, `http://p.example.com/hc/s.coap.example.com/foo` could either represent the target `coap://s.coap.example.com/foo` or `coaps://s.coap.example.com/foo` depending on application specific presets.

5.2.2. Encoding Caveats

When the authority of the Target CoAP URI is given as an IPv6address, then the surrounding square brackets MUST be percent-encoded in the Hosting HTTP URI, in order to comply with the syntax defined in Section 3.3. of [RFC3986] for a URI path segment. E.g.: `coap://[2001:db8::1]/light?on` becomes `http://p.example.com/hc/coap://%5B2001:db8::1%5D/light?on`.

Everything else can be safely copied verbatim from the Target CoAP URI to the Hosting HTTP URI.

5.3. URI Mapping Template

This section defines a format for the URI template [RFC6570] used by a reverse HC proxy to inform its clients about the expected syntax for the Hosting HTTP URI. This will then be used by the HTTP client to construct the URI to be sent in the HTTP request to the reverse HC proxy.

When instantiated, an URI Mapping Template is always concatenated to a HC Proxy URI provided by the reverse HC proxy via discovery (see Section 5.4), or by other means.

A simple form (Section 5.3.1) and an enhanced form (Section 5.3.2) are provided to fit different users' requirements.

Both forms are expressed as level 2 URI templates [RFC6570] to take care of the expansion of values that are allowed to include reserved URI characters. The syntax of all URI formats is specified in this section in Augmented Backus-Naur Form (ABNF) [RFC5234].

5.3.1. Simple Form

The simple form MUST be used for mappings where the Target CoAP URI is going to be copied (using rules of Section 5.2.2) at some fixed position into the Hosting HTTP URI.

The following template variables MUST be used in mutual exclusion in a template definition:

```
cu = coap-URI    ; from [RFC7252], Section 6.1
su = coaps-URI   ; from [RFC7252], Section 6.2
tu = cu / su
```

The same considerations as in Section 5.2.1 apply, in that the CoAP scheme may be omitted from the Hosting HTTP URI.

5.3.1.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use `http://p.example.com/hc` as the HC Proxy URI. Note that these examples all define mapping templates that deviate from the default template of Section 5.2 to be able to illustrate the use of the above template variables.

1. "coap" URI is a query argument of the Hosting HTTP URI:

?coap_target_uri={+cu}

coap://s.example.com/light

http://p.example.com/hc?coap_target_uri=coap://s.example.com/light

2. "coaps" URI is a query argument of the Hosting HTTP URI:

?coaps_target_uri={+su}

coaps://s.example.com/light

http://p.example.com/hc?coaps_target_uri=coaps://s.example.com/light

3. Target CoAP URI as a query argument of the Hosting HTTP URI:

?target_uri={+tu}

coap://s.example.com/light

http://p.example.com/hc?target_uri=coap://s.example.com/light

or

coaps://s.example.com/light

http://p.example.com/hc?target_uri=coaps://s.example.com/light

4. Target CoAP URI in the path component of the Hosting HTTP URI
(i.e., the default URI Mapping template):

/ {+tu}

coap://s.example.com/light

http://p.example.com/hc/coap://s.example.com/light

or

coaps://s.example.com/light

http://p.example.com/hc/coaps://s.example.com/light

5. "coap" URI is a query argument of the Hosting HTTP URI; client decides to omit scheme because a default scheme is agreed beforehand between client and proxy:

```
?coap_uri={+cu}
```

```
coap://s.example.com/light
```

```
http://p.example.com/hc?coap_uri=s.example.com/light
```

5.3.2. Enhanced Form

The enhanced form can be used to express more sophisticated mappings, i.e., those that do not fit into the simple form.

There MUST be at most one instance of each of the following template variables in a template definition:

```
s  = "coap" / "coaps" ; from [RFC7252], Sections 6.1 and 6.2
hp = host [":" port]  ; from [RFC3986] Sections 3.2.2 and 3.2.3
p  = path-abempty     ; from [RFC3986] Section 3.3
q  = query             ; from [RFC3986] Section 3.4
qq = [ "?" query ]    ; qq is empty iff 'query' is empty
```

5.3.2.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use `http://p.example.com/hc` as the HC Proxy URI.

1. Target CoAP URI components in path segments, and optional query in query component:

```
{+s}{+hp}{+p}{+qq}  
coap://s.example.com/light  
http://p.example.com/hc/coap/s.example.com/light  
or  
coap://s.example.com/light?on  
http://p.example.com/hc/coap/s.example.com/light?on
```

2. Target CoAP URI components split in individual query arguments:

```
?s={+s}&hp={+hp}&p={+p}&q={+q}  
coap://s.example.com/light  
http://p.example.com/hc?s=coap&hp=s.example.com&p=/light&q=  
or  
coaps://s.example.com/light?on  
http://p.example.com/hc?s=coaps&hp=s.example.com&p=/light&q=on
```

5.4. Discovery

In order to accommodate site specific needs while allowing third parties to discover the proxy function, the reverse HC proxy SHOULD publish information related to the location and syntax of the reverse HC proxy function using the CoRE Link Format [RFC6690] interface.

To this aim a new Resource Type, "core.hc", is defined in this document. It can be used as the value for the "rt" attribute in a query to the /.well-known/core in order to locate the URI where the reverse HC proxy function is anchored, i.e. the HC Proxy URI.

Along with it, the new target attribute "hct" is defined in this document. This attribute MAY be returned in a "core.hc" link to provide the URI Mapping Template associated to the mapping resource. The default template given in Section 5.2, i.e., {+tu}, MUST be assumed if no "hct" attribute is found in the returned link. If a "hct" attribute is present in the returned link, then a compliant client MUST use it to create the Hosting HTTP URI.

Discovery as specified in [RFC6690] SHOULD be available on both the HTTP and the CoAP side of the reverse HC proxy, with one important difference: on the CoAP side the link associated to the "core.hc" resource needs an explicit anchor referring to the HTTP origin, while on the HTTP interface the link context is already the HTTP origin carried in the request's Host header, and doesn't have to be made explicit.

5.4.1. Examples

- o The first example exercises the CoAP interface, and assumes that the default template, {+tu}, is used. For example, in use case #3 in section Section 4, the smartphone may discover the public reverse HC proxy before leaving the home network. Then when outside the home network, the smartphone will be able to query the appropriate home sensor.

```
Req:  GET coap://[ff02::1]/.well-known/core?rt=core.hc
```

```
Res:  2.05 Content
      </hc>;anchor="http://p.example.com";rt="core.hc"
```

- o The second example - also on the CoAP side of the reverse HC proxy - uses a custom template, i.e., one where the CoAP URI is carried inside the query component, thus the returned link carries the URI template to be used in an explicit "hct" attribute:

```
Req:  GET coap://[ff02::1]/.well-known/core?rt=core.hc
```

```
Res:  2.05 Content
      </hc>;anchor="http://p.example.com";
      rt="core.hc";hct="?uri={+tu}"
```

On the HTTP side, link information can be serialized in more than one way:

- o using the 'application/link-format' content type:

Req: GET /.well-known/core?rt=core.hc HTTP/1.1
Host: p.example.com

Res: HTTP/1.1 200 OK
Content-Type: application/link-format
Content-Length: 18

</hc>;rt="core.hc"

- o using the 'application/link-format+json' content type as defined in [I-D.ietf-core-links-json]:

Req: GET /.well-known/core?rt=core.hc HTTP/1.1
Host: p.example.com

Res: HTTP/1.1 200 OK
Content-Type: application/link-format+json
Content-Length: 31

[{"href":"/hc","rt":"core.hc"}]

- o using the Link header:

Req: GET /.well-known/core?rt=core.hc HTTP/1.1
Host: p.example.com

Res: HTTP/1.1 200 OK
Link: </hc>;rt="core.hc"

- o A reverse HC proxy may expose two different HC Proxy URIs to differentiate between Target CoAP resources in the "coap" and "coaps" scheme:

```
Req:  GET /.well-known/core?rt=core.hc
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format+json
      Content-Length: 111

      [
        {"href":"/hc/plaintext","rt":"core.hc","hct":"+cu"},
        {"href":"/hc/secure","rt":"core.hc","hct":"+su"}
      ]
```

6. Media Type Mapping

6.1. Overview

A reverse HC proxy needs to translate HTTP media types (Section 3.1.1.1 of [RFC7231]) and content encodings (Section 3.1.2.2 of [RFC7231]) into CoAP content formats (Section 12.3 of [RFC7252]) and vice versa.

Media type translation can happen in GET, PUT or POST requests going from HTTP to CoAP, and in 2.xx (i.e., successful) responses going from CoAP to HTTP. Specifically, PUT and POST need to map both the Content-Type and Content-Encoding HTTP headers into a single CoAP Content-Format option, whereas GET needs to map Accept and Accept-Encoding HTTP headers into a single CoAP Accept option. To generate the HTTP response, the CoAP Content-Format option is mapped back to a suitable HTTP Content-Type and Content-Encoding combination.

An HTTP request carrying a Content-Type and Content-Encoding combination which the reverse HC proxy is unable to map to an equivalent CoAP Content-Format, SHALL elicit a 415 (Unsupported Media Type) response by the reverse HC proxy.

On the content negotiation side, failure to map Accept and Accept-* headers SHOULD be silently ignored: the reverse HC proxy SHOULD therefore forward as a CoAP request with no Accept option. The reverse HC proxy thus disregards the Accept/Accept-* header fields by treating the response as if it is not subject to content negotiation, as mentioned in Sections 5.3.* of [RFC7231]. However, a reverse HC proxy implementation is free to attempt mapping a single Accept header in a GET request to multiple CoAP GET requests, each with a single Accept option, which are then tried in sequence until one succeeds. Note that an HTTP Accept */* MUST be mapped to a CoAP request without Accept option.

While the CoAP to HTTP direction has always a well defined mapping (with the exception examined in Section 6.2), the HTTP to CoAP direction is more problematic because the source set, i.e., potentially 1000+ IANA registered media types, is much bigger than the destination set, i.e., the mere 6 values initially defined in Section 12.3 of [RFC7252].

Depending on the tight/loose coupling with the application(s) for which it proxies, the reverse HC proxy could implement different media type mappings.

When tightly coupled, the reverse HC proxy knows exactly which content formats are supported by the applications, and can be strict when enforcing its forwarding policies in general, and the media type mapping in particular.

On the other side, when the reverse HC proxy is a general purpose application layer gateway, being too strict could significantly reduce the amount of traffic that it'd be able to successfully forward. In this case, the "loose" media type mapping detailed in Section 6.3 MAY be implemented.

The latter grants more evolution of the surrounding ecosystem, at the cost of allowing more attack surface. In fact, as a result of such strategy, payloads would be forwarded more liberally across the unconstrained/constrained network boundary of the communication path. Therefore, when applied, other forms of access control must be set in place to avoid unauthorized users to deplete or abuse systems and network resources.

6.2. 'application/coap-payload' Media Type

If the reverse HC proxy receives a CoAP response with a Content-Format that it does not recognize (e.g. because the value has been registered after the proxy has been deployed, or the CoAP server uses an experimental value which is not registered), then the reverse HC proxy SHALL return a generic "application/coap-payload" media type with numeric parameter "cf" as defined in Section 9.2.

For example, the CoAP content format '60' ("application/cbor") would be represented by "application/coap-payload;cf=60", would '60' be an unknown content format to the reverse HC Proxy.

A HTTP client MAY use the media type "application/coap-payload" as a means to send a specific content format to a CoAP server via a reverse HC Proxy if the client has determined that the reverse HC Proxy does not directly support the type mapping it needs. This case

may happen when dealing for example with newly registered, yet to be registered, or experimental CoAP content formats.

6.3. Loose Media Type Mapping

By structuring the type information in a super-class (e.g. "text") followed by a finer grained sub-class (e.g. "html"), and optional parameters (e.g. "charset=utf-8"), Internet media types provide a rich and scalable framework for encoding the type of any given entity.

This approach is not applicable to CoAP, where Content Formats conflate an Internet media type (potentially with specific parameters) and a content encoding into one small integer value.

To remedy this loss of flexibility, we introduce the concept of a "loose" media type mapping, where media types that are specializations of a more generic media type can be aliased to their super-class and then mapped (if possible) to one of the CoAP content formats. For example, "application/soap+xml" can be aliased to "application/xml", which has a known conversion to CoAP. In the context of this "loose" media type mapping, "application/octet-stream" can be used as a fallback when no better alias is found for a specific media type.

Table 1 defines the default lookup table for the "loose" media type mapping. Given an input media type, the table returns its best generalized media type using the most specific match i.e. the table entries are compared to the input in top to bottom order until an entry matches.

Internet media type	Generalized media type
application/*+xml	application/xml
application/*+json	application/json
text/xml	application/xml
text/*	text/plain
/	application/octet-stream

Table 1: Media type generalization lookup table

The "loose" media type mapping is an OPTIONAL feature. Implementations supporting this kind of mapping SHOULD provide a flexible way to define the set of media type generalizations allowed.

6.4. Media Type to Content Format Mapping Algorithm

This section defines the algorithm used to map an HTTP Internet media type to its correspondent CoAP content format.

The algorithm uses the mapping table defined in Section 12.3 of [RFC7252] plus, possibly, any locally defined extension of it. Optionally, the table and lookup mechanism described in Section 6.3 can be used if the implementation chooses so.

Note that the algorithm may have side effects on the associated representation (see also Section 6.5).

In the following:

- o C-T, C-E, and C-F stand for the values of the Content-Type (or Accept) HTTP header, Content-Encoding (or Accept-Encoding) HTTP header, and Content-Format CoAP option respectively.
- o If C-E is not given it is assumed to be "identity".
- o MAP is the mandatory lookup table, GMAP is the optional generalized table.

INPUT: C-T and C-E
OUTPUT: C-F or Fail

```
1.  if no C-T: return Fail
2.  C-F = MAP[C-T, C-E]
3.  if C-F is not None: return C-F
4.  if C-E is not "identity":
5.    if C-E is supported (e.g. gzip):
6.      decode the representation accordingly
7.      set C-E to "identity"
8.    else:
9.      return Fail
10. repeat steps 2. and 3.
11. if C-T allows a non-lossy transformation into \
12.   one of the supported C-F:
13.   transcode the representation accordingly
14.   return C-F
15. if GMAP is defined:
16.   C-F = GMAP[C-T]
17.   if C-F is not None: return C-F
18. return Fail
```

Figure 2

6.5. Content Transcoding

6.5.1. General

Payload content transcoding (e.g. see steps 11-14 of Figure 2) is an OPTIONAL feature. Implementations supporting this feature should provide a flexible way to define the set of transcodings allowed.

As noted in Section 6.4, the process of mapping the media type can have side effects on the forwarded entity body. This may be caused by the removal or addition of a specific content encoding, or because the reverse HC proxy decides to transcode the representation to a different (compatible) format. The latter proves useful when an optimized version of a specific format exists. For example an XML-encoded resource could be transcoded to Efficient XML Interchange (EXI) format, or a JSON-encoded resource into CBOR [RFC7049], effectively achieving compression without losing any information.

However, it should be noted that in certain cases, transcoding can lose information in a non-obvious manner. For example, encoding an XML document using schema-informed EXI encoding leads to a loss of information when the destination does not know the exact schema version used by the encoder, which means that whenever the reverse HC proxy transcodes an application/XML to application/EXI in-band metadata could be lost. Therefore, the implementer should always carefully verify such lossy payload transformations before triggering the transcoding.

6.5.2. CoRE Link Format

The CoRE Link Format [RFC6690] is a set of links (i.e., URIs and their formal relationships) which is carried as content payload in a CoAP response. These links usually include CoAP URIs that might be translated by the reverse HC proxy to the correspondent HTTP URIs using the implemented URI mapping function (see Section 5). Such a process would inspect the forwarded traffic and attempt to re-write the body of resources with an application/link-format media type, mapping the embedded CoAP URIs to their HTTP counterparts. Some potential issues with this approach are:

1. The client may be interested to retrieve original (unaltered) CoAP payloads through the reverse HC proxy, not modified versions.
2. Tampering with payloads is incompatible with resources that are integrity protected (although this is a problem with transcoding in general).

3. The reverse HC proxy needs to fully understand [RFC6690] syntax and semantics, otherwise there is an inherent risk to corrupt the payloads.

Therefore, CoRE Link Format payload should only be transcoded at the risk and discretion of the proxy implementer.

6.5.3. Diagnostic Messages

CoAP responses may, in certain error cases, contain a diagnostic message in the payload explaining the error situation, as described in Section 5.5.2 of [RFC7252]. If present, the CoAP response diagnostic payload SHOULD be copied in the HTTP response body. The CoAP diagnostic message MUST NOT be copied into the HTTP reason-phrase, since it potentially contains CR-LF characters which are incompatible with HTTP reason-phrase syntax.

7. Response Code Mapping

Table 2 defines the HTTP response status codes to which each CoAP response code SHOULD be mapped. This table complies with the requirements in Section 10.2 of [RFC7252] and is intended to cover all possible cases. Multiple appearances of a HTTP status code in the second column indicates multiple equivalent HTTP responses are possible based on the same CoAP response code, depending on the conditions cited in the Notes (third column and text below table).

CoAP Response Code	HTTP Status Code	Notes
2.01 Created	201 Created	1
2.02 Deleted	200 OK	2
	204 No Content	2
2.03 Valid	304 Not Modified	3
	200 OK	4
2.04 Changed	200 OK	2
	204 No Content	2
2.05 Content	200 OK	
4.00 Bad Request	400 Bad Request	
4.01 Unauthorized	403 Forbidden	5
4.02 Bad Option	400 Bad Request	6
4.02 Bad Option	500 Internal Server Error	6
4.03 Forbidden	403 Forbidden	
4.04 Not Found	404 Not Found	
4.05 Method Not Allowed	400 Bad Request	7
4.06 Not Acceptable	406 Not Acceptable	
4.12 Precondition Failed	412 Precondition Failed	
4.13 Request Ent. Too Large	413 Request Repr. Too Large	
4.15 Unsupported Media Type	415 Unsupported Media Type	
5.00 Internal Server Error	500 Internal Server Error	
5.01 Not Implemented	501 Not Implemented	
5.02 Bad Gateway	502 Bad Gateway	
5.03 Service Unavailable	503 Service Unavailable	8
5.04 Gateway Timeout	504 Gateway Timeout	
5.05 Proxying Not Supported	502 Bad Gateway	9

Table 2: CoAP-HTTP Response Code Mappings

Notes:

1. A CoAP server may return an arbitrary format payload along with this response. This payload SHOULD be returned as entity in the HTTP 201 response. Section 7.3.2 of [RFC7231] does not put any requirement on the format of the entity. (In the past, [RFC2616] did.)
2. The HTTP code is 200 or 204 respectively for the case that a CoAP server returns a payload or not. [RFC7231] Section 5.3 requires code 200 in case a representation of the action result is returned for DELETE/POST/PUT, and code 204 if not. Hence, a proxy SHOULD transfer any CoAP payload contained in a CoAP 2.02 response to the HTTP client using a 200 OK response.

3. HTTP code 304 (Not Modified) is sent if the HTTP client performed a conditional HTTP request and the CoAP server responded with 2.03 (Valid) to the corresponding CoAP validation request. Note that Section 4.1 of [RFC7232] puts some requirements on header fields that must be present in the HTTP 304 response.
4. A 200 response to a CoAP 2.03 occurs only when the reverse HC proxy, for efficiency reasons, is running a local cache. An unconditional HTTP GET which produces a cache-hit, could trigger a re-validation (i.e. a conditional GET) on the CoAP side. The proxy receiving 2.03 updates the freshness of its cached representation and returns it to the HTTP client.
5. A HTTP 401 Unauthorized (Section 3.1 of [RFC7235]) response is not applicable because there is no equivalent in CoAP of WWW-Authenticate which is mandatory in a HTTP 401 response.
6. If the proxy has a way to determine that the Bad Option is due to the straightforward mapping of a client request header into a CoAP option, then returning HTTP 400 (Bad Request) is appropriate. In all other cases, the proxy MUST return HTTP 500 (Internal Server Error) stating its inability to provide a suitable translation to the client's request.
7. A CoAP 4.05 (Method Not Allowed) response SHOULD normally be mapped to a HTTP 400 (Bad Request) code, because the HTTP 405 response would require specifying the supported methods - which are generally unknown. In this case the reverse HC Proxy SHOULD also return a HTTP reason-phrase in the HTTP status line that starts with the string "CoAP server returned 4.05" in order to facilitate troubleshooting. However, if the reverse HC proxy has more granular information about the supported methods for the requested resource (e.g. via a Resource Directory ([I-D.ietf-core-resource-directory])) then it MAY send back a HTTP 405 (Method Not Allowed) with a properly filled in "Allow" response-header field (Section 7.4.1 of [RFC7231]).
8. The value of the HTTP "Retry-After" response-header field is taken from the value of the CoAP Max-Age Option, if present.
9. This CoAP response can only happen if the proxy itself is configured to use a CoAP forward-proxy (Section 5.7 of [RFC7252]) to execute some, or all, of its CoAP requests.

8. Additional Mapping Guidelines

8.1. Caching and Congestion Control

A reverse HC proxy SHOULD cache CoAP responses and reply, whenever applicable, with a cached representation of the requested resource.

If the HTTP client times out and drops the HTTP session to the reverse HC proxy (closing the TCP connection) after the HTTP request was made, a reverse HC proxy SHOULD wait for the associated CoAP response and cache it if possible. Subsequent requests to the reverse HC proxy for the same resource can use the result present in cache, or, if a response has still to come, the HTTP requests will wait on the open CoAP request.

According to [RFC7252], a proxy must limit the number of outstanding interactions to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the reverse HC proxy SHOULD also pose a limit to the number of concurrent CoAP requests pending on the same constrained network; further incoming requests MAY either be queued or dropped (returning 503 Service Unavailable). This limit and the proxy queueing/dropping behavior SHOULD be configurable. In order to effectively apply above congestion control, the reverse HC proxy should be server-side placed.

Resources experiencing a high access rate coupled with high volatility MAY be observed [RFC7641] by the reverse HC proxy to keep their cached representation fresh while minimizing the number of CoAP traffic in the constrained network. See Section 8.2.

8.2. Cache Refresh via Observe

There are cases where using the CoAP observe protocol [RFC7641] to handle proxy cache refresh is preferable to the validation mechanism based on ETag as defined in [RFC7252]. Such scenarios include, but are not limited to, sleepy CoAP nodes -- with possibly high variance in requests' distribution -- which would greatly benefit from a server driven cache update mechanism. Ideal candidates for CoAP observe are also crowded or very low throughput networks, where reduction of the total number of exchanged messages is an important requirement.

This subsection aims at providing a practical evaluation method to decide whether refreshing a cached resource R is more efficiently handled via ETag validation or by establishing an observation on R.

Let T_R be the mean time between two client requests to resource R, let T_C be the mean time between two representation changes of R, and

let M_R be the mean number of CoAP messages per second exchanged to and from resource R. If we assume that the initial cost for establishing the observation is negligible, an observation on R reduces M_R iff $T_R < 2 \cdot T_C$ with respect to using ETag validation, that is iff the mean arrival rate of requests for resource R is greater than half the change rate of R.

When observing the resource R, M_R is always upper bounded by $2/T_C$.

8.3. Use of CoAP Blockwise Transfer

A reverse HC proxy SHOULD support CoAP blockwise transfers [I-D.ietf-core-block] to allow transport of large CoAP payloads while avoiding excessive link-layer fragmentation in constrained networks, and to cope with small datagram buffers in CoAP end-points as described in [RFC7252] Section 4.6.

A reverse HC proxy SHOULD attempt to retry a payload-carrying CoAP PUT or POST request with blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. A reverse HC proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than BLOCKWISE_THRESHOLD bytes. The value of BLOCKWISE_THRESHOLD is implementation-specific, for example it can be:

- o calculated based on a known or typical UDP datagram buffer size for CoAP end-points, or
- o set to N times the known size of a link-layer frame in a constrained network where e.g. $N=5$, or
- o preset to a known IP MTU value, or
- o set to a known Path MTU value.

The value BLOCKWISE_THRESHOLD, or the parameters from which it is calculated, should be configurable in a proxy implementation. The maximum block size the proxy will attempt to use in CoAP requests should also be configurable.

The reverse HC proxy SHOULD detect CoAP end-points not supporting blockwise transfers by checking for a 4.02 (Bad Option) response returned by an end-point in response to a CoAP request with a Block* Option, and subsequent absence of the 4.02 in response to the same request without Block* Options. This allows the reverse HC proxy to be more efficient, not attempting repeated blockwise transfers to CoAP servers that do not support it. However, if a request payload is too large to be sent as a single CoAP request and blockwise

transfer would be unavoidable, the proxy still SHOULD attempt blockwise transfer on such an end-point before returning the response 413 (Request Entity Too Large) to the HTTP client.

For improved latency an reverse HC proxy MAY initiate a blockwise CoAP request triggered by an incoming HTTP request even when the HTTP request message has not yet been fully received, but enough data has been received to send one or more data blocks to a CoAP server already. This is particularly useful on slow client-to-proxy connections.

8.4. Security Translation

For the guidelines on security context translations for a reverse HC proxy, see Section 10.2. A translation may involve e.g. applying a rule that any "https" request is translated to a "coaps" request, or e.g. applying a rule that a "https" request is translated to an unsecured "coap" request.

8.5. CoAP Multicast

A reverse HC proxy MAY support CoAP multicast. If it does, the reverse HC proxy sends out a multicast CoAP request if the Target CoAP URI's authority is a multicast IP literal or resolves to a multicast IP address; assuming the proper security measures are in place to mitigate security risks of CoAP multicast (Section 10). If the security policies do not allow the specific CoAP multicast request to be made, the reverse HC proxy SHOULD respond 403 (Forbidden).

If a reverse HC proxy does not support CoAP multicast, it SHOULD respond 403 (Forbidden) to any valid HTTP request that maps to a CoAP multicast request.

Details related to supporting CoAP multicast are currently out of scope of this document since in a reverse proxy scenario a HTTP client typically expects to receive a single response, not multiple. However, a reverse HC proxy that implements CoAP multicast MAY include application-specific functions to aggregate multiple CoAP responses into a single HTTP response. We suggest using the "application/http" internet media type (Section 8.3.2 of [RFC7230]) to enclose a set of one or more HTTP response messages, each representing the mapping of one CoAP response.

8.6. Timeouts

When facing long delays of a CoAP server in responding, the HTTP client or any other proxy in between MAY timeout. Further discussion of timeouts in HTTP is available in Section 6.2.4 of [RFC7230].

A reverse HC proxy MUST define an internal timeout for each pending CoAP request, because the CoAP server may silently die before completing the request. Assuming the Proxy may use confirmable CoAP requests, such timeout value T SHOULD be at least

$$T = \text{MAX_RTT} + \text{MAX_SERVER_RESPONSE_DELAY}$$

where MAX_RTT is defined in [RFC7252] and MAX_SERVER_RESPONSE_DELAY is defined in [RFC7390]. An exception to this rule occurs when the reverse HC proxy is configured with a HTTP response timeout value that is lower than above value T; then the lower value should be also used as the CoAP request timeout.

8.7. Miscellaneous

In certain use cases, constrained CoAP nodes do not make use of the DNS protocol. However even when the DNS protocol is not used in a constrained network, defining valid FQDN (i.e., DNS entries) for constrained CoAP servers, where possible, may help HTTP clients to access the resources offered by these servers via a reverse HC proxy.

HTTP connection pipelining (section 6.3.2 of [RFC7230]) may be supported by a reverse HC proxy. This is transparent to the CoAP servers: the reverse HC proxy will serve the pipelined requests by issuing different CoAP requests. The reverse HC proxy in this case needs to respect the NSTART limit of Section 4.7 of [RFC7252].

9. IANA Considerations

9.1. New 'core.hc' Resource Type

This document registers a new Resource Type (rt=) Link Target Attribute, 'core.hc', in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

Attribute Value: core.hc

Description: HTTP to CoAP mapping base resource.

Reference: See Section 5.4.

9.2. New 'coap-payload' Internet Media Type

This document defines the "application/coap-payload" media type with a single parameter "cf". This media type represents any payload that a CoAP message can carry, having a content format that can be identified by a CoAP Content-Format parameter (an integer in range 0-65535). The parameter "f" is the integer defining the CoAP content format.

Type name: application

Subtype name: coap-payload

Required parameters:

cf - CoAP Content-Format integer in range 0-65535 denoting the content format of the CoAP payload carried.

Optional parameters: None

Encoding considerations:

The specific CoAP content format encoding considerations for the selected Content-Format (cf parameter) apply.

Security considerations:

The specific CoAP content format security considerations for the selected Content-Format (cf parameter) apply.

Interoperability considerations:

Published specification: (this I-D - TBD)

Applications that use this media type:

HTTP-to-CoAP Proxies.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information:

Esko Dijk ("esko@ieee.org")

Intended usage: COMMON

Restrictions on usage:

An application (or user) can only use this media type if it has to represent a CoAP payload of which the specified CoAP Content-Format is an unrecognized number; such that a proper translation directly to the equivalent HTTP media type is not possible.

Author: CoRE WG

Change controller: IETF

Provisional registration? (standards tree only): N/A

10. Security Considerations

The security concerns raised in Section 9.2 of [RFC7230] also apply to the reverse HC proxy scenario. In fact, the reverse HC proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the reverse HC proxy cannot be dropped, because the protocol translation function is the core duty of the reverse HC proxy: it is a necessarily trusted, impossible to bypass, component in the communication path.

A reverse proxy deployed at the boundary of a constrained network is an easy single point of failure for reducing availability. As such, special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it has fewer limitations than the constrained devices it is serving.

The following sub paragraphs categorize and discuss a set of specific security issues related to the translation, caching and forwarding functionality exposed by a reverse HC proxy.

10.1. Traffic Overflow

Due to the typically constrained nature of CoAP nodes, particular attention SHOULD be given to the implementation of traffic reduction mechanisms (see Section 8.1), because inefficient proxy

implementations can be targeted by unconstrained Internet attackers. Bandwidth or complexity involved in such attacks is very low.

An amplification attack to the constrained network may be triggered by a multicast request generated by a single HTTP request which is mapped to a CoAP multicast resource, as considered in Section 11.3 of [RFC7252].

The risk likelihood of this amplification technique is higher than an amplification attack carried out by a malicious constrained device (e.g. ICMPv6 flooding, like Packet Too Big, or Parameter Problem on a multicast destination [RFC4732]), since it does not require direct access to the constrained network.

The feasibility of this attack, disruptive in terms of CoAP server availability, can be limited by access controlling the exposed HTTP multicast resources, so that only known/authorized users access such URIs.

10.2. Handling Secured Exchanges

An HTTP request can be sent to the reverse HC proxy over a secured connection. However, there may not always exist a secure connection mapping to CoAP. For example, a secure distribution method for multicast traffic is complex and MAY not be implemented (see [RFC7390]).

A reverse HC proxy SHOULD implement explicit rules for security context translations. A translation may involve e.g. applying a rule that any "https" unicast request is translated to a "coaps" request, or e.g. applying a rule that a "https" request is translated to an unsecured "coap" request. Another rule could specify the security policy and parameters used for DTLS connections. Such rules will largely depend on the application and network context in which a proxy operates. These rules SHOULD be configurable in a reverse HC proxy.

If a policy for access to 'coaps' URIs is configurable in a reverse HC proxy, it is RECOMMENDED that the policy is by default configured to disallow access to any 'coaps' URI by a HTTP client using an unsecured (non-TLS) connection. Naturally, a user MAY reconfigure the policy to allow such access in specific cases.

By default, a reverse HC proxy SHOULD reject any secured client request if there is no configured security policy mapping. This recommendation MAY be relaxed in case the destination network is believed to be secured by other, complementary, means. E.g.: assumed that CoAP nodes are isolated behind a firewall (e.g. as in the

server-side reverse HC proxy deployment shown in Figure 1), the reverse HC proxy may be configured to translate the incoming HTTPS request using plain CoAP (NoSec mode).

The HTTP-CoAP URI mapping (defined in Section 5) MUST NOT map to HTTP a CoAP resource intended to be only accessed securely.

A secured connection that is terminated at the reverse HC proxy, i.e., the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The reverse HC proxy SHOULD NOT cache any secured content to avoid any leak of secured information. However, in some specific scenario, a security/efficiency trade-off could motivate caching secured information; in that case the caching behavior MAY be tuned to some extent on a per-resource basis.

10.3. Proxy and CoAP Server Resource Exhaustion

If the reverse HC proxy implements the low-latency optimization of Section 8.3 intended for slow client-to-proxy connections, the Proxy may become vulnerable to a resource exhaustion attack. In this case an attacking client could initiate multiple requests using a relatively large message body which is (after an initial fast transfer) transferred very slowly to the Proxy. This would trigger the reverse HC proxy to create state for a blockwise CoAP request per HTTP request, waiting for the arrival of more data over the HTTP/TCP connection. Such attacks can be mitigated in the usual ways for HTTP servers using for example a connection time limit along with a limit on the number of open TCP connections per IP address.

10.4. URI Mapping

The following risks related to the URI mapping described in Section 5 and its use by HC proxies have been identified:

DoS attack on the constrained/CoAP network.

To mitigate, by default deny any Target CoAP URI whose authority is (or maps to) a multicast address. Then explicitly white-list multicast resources/authorities that are allowed to be de-referenced. See also Section 8.5.

Leaking information on the constrained/CoAP network resources and topology.

To mitigate, by default deny any Target CoAP URI (especially /.well-known/core is a resource to be protected), and then explicit white-list resources that are allowed to be seen from outside.

Reduced privacy due to the mechanics of the URI mapping.

The internal CoAP Target resource is totally transparent from outside. A reverse HC proxy can mitigate by implementing a HTTPS-only interface, making the Target CoAP URI totally opaque to a passive attacker.

11. Acknowledgements

An initial version of Table 2 in Section 7 has been provided in revision -05 of the CoRE CoAP I-D. Special thanks to Peter van der Stok for countless comments and discussions on this document, that contributed to its current structure and text.

Thanks to Carsten Bormann, Zach Shelby, Michele Rossi, Nicola Bui, Michele Zorzi, Klaus Hartke, Cullen Jennings, Kepeng Li, Brian Frank, Peter Saint-Andre, Kerry Lynn, Linyi Tian, Dorothy Gellert, Francesco Corazza for helpful comments and discussions that have shaped the document.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n.251557.

12. References

12.1. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.

- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<http://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<http://www.rfc-editor.org/info/rfc7231>>.
- [RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <<http://www.rfc-editor.org/info/rfc7232>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <<http://www.rfc-editor.org/info/rfc7235>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

12.2. Informative References

- [I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-04 (work in progress), November 2015.

- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-05 (work in progress), October 2015.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <<http://www.rfc-editor.org/info/rfc2616>>.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, DOI 10.17487/RFC3040, January 2001, <<http://www.rfc-editor.org/info/rfc3040>>.
- [RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <<http://www.rfc-editor.org/info/rfc4732>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<http://www.rfc-editor.org/info/rfc7390>>.

Appendix A. Change Log

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-07 to ietf-08:

- o Addressed WGLC review comments from Klaus Hartke as per the correspondence of March 9, 2016 on the CORE WG mailing list.

Changes from ietf-06 to ietf-07:

- o Addressed Ticket #384 - Section 5.4.1 describes briefly (informative) how to discover CoAP resources from an HTTP client.
- o Addressed Ticket #378 - For HTTP media type to CoAP content format mapping and vice versa: a new draft (TBD) may be proposed in CoRE which describes an approach for automatic updating of the media

type mapping. This was noted in Section 6.1 but is otherwise outside the scope of this draft.

- o Addressed Ticket #377 - Added IANA section that defines a new HTTP media type "application/coap-payload" and created new Section 6.2 on how to use it.
- o Addressed Ticket #376 - Updated Table 2 (and corresponding note 7) to indicate that a CoAP 4.05 (Method Not Allowed) Response Code should be mapped to a HTTP 400 (Bad Request).
- o Added note to comply to ABNF when translating CoAP diagnostic payload to reason-phrase in Section 6.5.3.

Changes from ietf-05 to ietf-06:

- o Fully restructured the draft, bringing introductory text more to the front and allocating main sections to each of the key topics; addressing Ticket #379;
- o Addressed Ticket #382, fix of enhanced form URI template definition of q in Section 5.3.2;
- o Addressed Ticket #381, found a mapping 4.01 to 401 Unauthorized in Section 7;
- o Addressed Ticket #380 (Add IANA registration for "core.hc" Resource Type) in Section 9;
- o Addressed Ticket #376 (CoAP 4.05 response can't be translated to HTTP 405 by HC proxy) in Section 7 by use of empty 'Allow' header;
- o Removed details on the pros and cons of HC proxy placement options;
- o Addressed review comments of Carsten Bormann;
- o Clarified failure in mapping of HTTP Accept headers (Section 6.3);
- o Clarified detection of CoAP servers not supporting blockwise (Section 8.3);
- o Changed CoAP request timeout min value to MAX_RTT + MAX_SERVER_RESPONSE_DELAY (Section 8.6);
- o Added security section item (Section 10.3) related to use of CoAP blockwise transfers;

- o Many editorial improvements.

Changes from ietf-04 to ietf-05:

- o Addressed Ticket #366 (Mapping of CoRE Link Format payloads to be valid in HTTP Domain?) in Section 6.3.3.2 (Content Transcoding - CORE Link Format);
- o Addressed Ticket #375 (Add requirement on mapping of CoAP diagnostic payload) in Section 6.3.3.3 (Content Transcoding - Diagnostic Messages);
- o Addressed comment from Yusuke (<http://www.ietf.org/mail-archive/web/core/current/msg05491.html>) in Section 6.3.3.1 (Content Transcoding - General);
- o Various editorial improvements.

Changes from ietf-03 to ietf-04:

- o Expanded use case descriptions in Section 4;
- o Fixed/enhanced discovery examples in Section 5.4.1;
- o Addressed Ticket #365 (Add text on media type conversion by HTTP-CoAP proxy) in new Section 6.3.1 (Generalized media type mapping) and new Section 6.3.2 (Content translation);
- o Updated HTTPBis WG draft references to recently published RFC numbers.
- o Various editorial improvements.

Changes from ietf-02 to ietf-03:

- o Closed Ticket #351 "Add security implications of proposed default HTTP-CoAP URI mapping";
- o Closed Ticket #363 "Remove CoAP scheme in default HTTP-CoAP URI mapping";
- o Closed Ticket #364 "Add discovery of HTTP-CoAP mapping resource(s)".

Changes from ietf-01 to ietf-02:

- o Selection of single default URI mapping proposal as proposed to WG mailing list 2013-10-09.

Changes from ietf-00 to ietf-01:

- o Added URI mapping proposals to Section 4 as per the Email proposals to WG mailing list from Esko.

Authors' Addresses

Angelo P. Castellani
University of Padova
Via Gradenigo 6/B
Padova 35131
Italy

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal H3A 3G4
Canada

Phone: +1 514 585 0761
Email: Akbar.Rahman@InterDigital.com

Thomas Fossati
Alcatel-Lucent
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: thomas.fossati@alcatel-lucent.com

Esko Dijk
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
The Netherlands

Email: esko.dijk@philips.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 1, 2017

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
Nokia
E. Dijk
Philips Lighting
November 28, 2016

Guidelines for HTTP-to-CoAP Mapping Implementations
draft-ietf-core-http-mapping-17

Abstract

This document provides reference information for implementing a cross-protocol network proxy that performs translation from the HTTP protocol to CoAP (Constrained Application Protocol). This will enable an HTTP client to access resources on a CoAP server through the proxy. This document describes how an HTTP request is mapped to a CoAP request, and then how a CoAP response is mapped back to an HTTP response. This includes guidelines for status code, URI, and media type mappings, as well as additional interworking advice.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 1, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. HTTP-to-CoAP Proxy	5
4. Use Cases	6
5. URI Mapping	7
5.1. URI Terminology	8
5.2. Null Mapping	8
5.3. Default Mapping	8
5.3.1. Optional Scheme Omission	9
5.3.2. Encoding Caveats	9
5.4. URI Mapping Template	9
5.4.1. Simple Form	10
5.4.2. Enhanced Form	11
5.5. Discovery	13
5.5.1. Examples	13
6. Media Type Mapping	15
6.1. Overview	15
6.2. 'application/coap-payload' Media Type	16
6.3. Loose Media Type Mapping	17
6.4. Media Type to Content Format Mapping Algorithm	18
6.5. Content Transcoding	19
6.5.1. General	19
6.5.2. CoRE Link Format	20
6.5.3. Diagnostic Messages	20
7. Response Code Mapping	21
8. Additional Mapping Guidelines	23
8.1. Caching and Congestion Control	23
8.2. Cache Refresh via Observe	24
8.3. Use of CoAP Blockwise Transfer	24
8.4. CoAP Multicast	25
8.5. Timeouts	26

9. IANA Considerations	26
9.1. New 'core.hc' Resource Type	26
9.2. New 'coap-payload' Internet Media Type	26
10. Security Considerations	28
10.1. Multicast	29
10.2. Traffic Overflow	29
10.3. Handling Secured Exchanges	30
10.4. URI Mapping	30
11. Acknowledgments	31
12. References	31
12.1. Normative References	31
12.2. Informative References	33
Appendix A. Media Type Mapping Source Code	34
Appendix B. Change Log	38
Authors' Addresses	43

1. Introduction

CoAP (Constrained Application Protocol) [RFC7252] has been designed with the twofold aim to be an application protocol specialized for constrained environments and to be easily used in Representational State Transfer (REST) [Fielding] based architectures such as the Web. The latter goal has led to defining CoAP to easily interoperate with HTTP [RFC7230] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [RFC7252] describes the fundamentals of the CoAP-to-HTTP and the HTTP-to-CoAP cross-protocol mapping process. However, [RFC7252] focuses on the basic mapping of request methods and simple response code mapping between HTTP and CoAP, while leaving many details of the cross-protocol proxy for future definition. Therefore, a primary goal of this document is to define a consistent set of guidelines that an HTTP-to-CoAP proxy implementation should adhere to. The key benefit to adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability between an HTTP client and a CoAP server independent of the proxy that implements the cross-protocol mapping. (For example, a proxy conforming to these guidelines made by vendor A can be easily replaced by a proxy from vendor B that also conforms to the guidelines without breaking API semantics.)

This document describes HTTP mappings that apply to protocol elements defined in the base CoAP specification [RFC7252]. It is up to CoAP protocol extensions (new methods, response codes, options, content-formats) to describe their own HTTP mappings, if applicable.

The rest of this document is organized as follows:

- o Section 2 defines proxy terminology;
- o Section 3 introduces the HTTP-to-CoAP proxy;
- o Section 4 lists use cases in which HTTP clients need to contact CoAP servers;
- o Section 5 introduces a null, default, and advanced HTTP-to-CoAP URI mapping syntax;
- o Section 6 describes how to map HTTP media types to CoAP content formats and vice versa;
- o Section 7 describes how to map CoAP responses to HTTP responses;
- o Section 8 describes additional mapping guidelines related to caching, congestion, timeouts, etc.;
- o Section 10 discusses possible security impact of HTTP-to-CoAP protocol mapping.

2. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This specification requires readers to be familiar with the vocabulary and concepts discussed in [RFC7228], in particular, the terms "Constrained Nodes" and "Constrained Networks". In addition, this specification makes use of the following terms:

HC Proxy

A proxy performing a cross-protocol mapping, in the context of this document an HTTP-to-CoAP (HC) mapping. Specifically, the HC Proxy acts as an HTTP server and a CoAP client. The HC Proxy can take on the role of a Forward, Reverse or Interception Proxy.

Forward Proxy (or Forward HC Proxy)

A message forwarding agent that is selected by the HTTP client, usually via local configuration rules, to receive requests for some type(s) of absolute URI and to attempt to satisfy those requests via translation to the protocol indicated by the absolute URI. The user decides (is willing) to use the proxy as the forwarding/de-referencing agent for a predefined subset of the URI space. In [RFC7230] this is called a Proxy. [RFC7252] defines Forward-Proxy similarly.

Reverse Proxy (or Reverse HC Proxy)

As in [RFC7230], a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server's protocol. A Reverse HC Proxy behaves as an origin (HTTP) server on its connection from the HTTP client. The HTTP client uses the "origin-form" (Section 5.3.1 of [RFC7230]) as a request-target URI. (Note that a Reverse Proxy appears to an HTTP client as an origin server while a Forward Proxy does not. So, when communicating with a Reverse Proxy a client may be unaware it is communicating with a proxy at all.)

Interception Proxy (or Interception HC Proxy)

As in [RFC3040], a proxy that receives inbound HTTP traffic flows through the process of traffic redirection, transparent to the HTTP client.

3. HTTP-to-CoAP Proxy

An HC Proxy is accessed by an HTTP client that needs to fetch a resource on a CoAP server. The HC Proxy handles the HTTP request by mapping it to the equivalent CoAP request, which is then forwarded to the appropriate CoAP server. The received CoAP response is then mapped to an appropriate HTTP response and finally sent back to the originating HTTP client.

Section 10.2 of [RFC7252] defines basic normative requirements on HTTP-to-CoAP mapping. This document provides additional details and guidelines for the implementation of an HC Proxy.

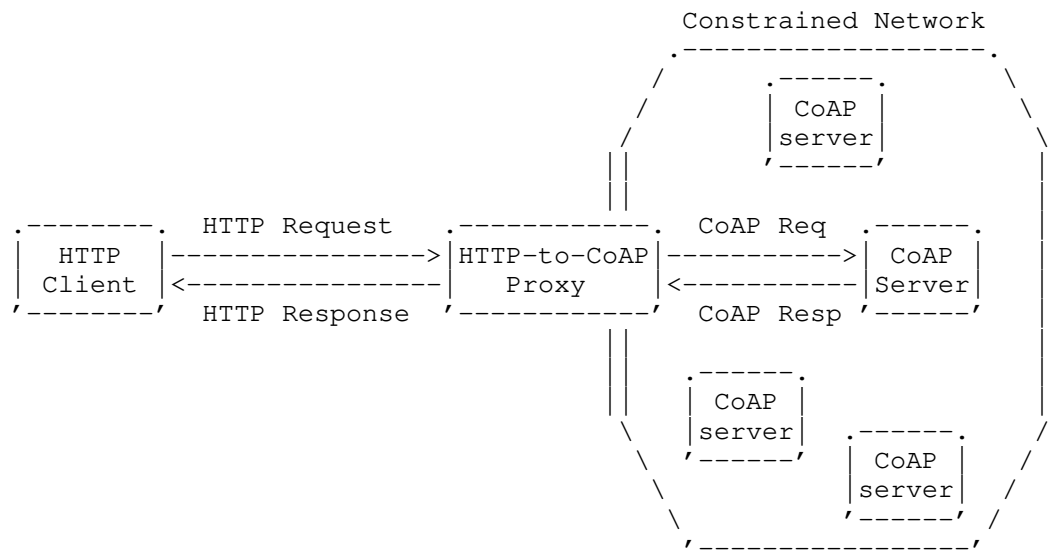


Figure 1: HTTP-To-CoAP Proxy Deployment Scenario

Figure 1 illustrates an example deployment scenario. There, an HC Proxy is located at the boundary of the Constrained Network domain and acts as an Application Layer Gateway (ALG) that allows only a very specific type of traffic (i.e., authorized inbound HTTP requests and their associated outbound CoAP responses) to pass through. All other kinds of traffic are segregated within the respective network segments.

4. Use Cases

To illustrate a few situations in which HTTP to CoAP protocol translation may be used, three use cases are described below.

1. Legacy building control application without CoAP: A building control application that uses HTTP but not CoAP can check the status of CoAP sensors and/or control actuators via an HC Proxy.
2. Making sensor data available to 3rd parties on the Web: For demonstration or public interest purposes, an HC Proxy may be configured to expose the contents of a CoAP sensor to the world via the web (HTTP and/or HTTPS). Some sensors may only accept secure 'coaps' requests, therefore the proxy is configured to translate requests to those devices accordingly. The HC Proxy is furthermore configured to only pass through GET requests in order to protect the constrained network.

3. Smartphone and home sensor: A smartphone can access directly a CoAP home sensor using a mutually authenticated 'https' request, provided its home router runs an HC Proxy and is configured with the appropriate certificate. An HTML5 [W3C.REC-html5-20141028] application on the smartphone can provide a friendly UI using the standard (HTTP) networking functions of HTML5.

A key point in the above use cases is the expected nature of the URI to be used by the HTTP client initiating the HTTP request to the HC Proxy. Specifically, in use case #1, there will be no 'coap' or 'coaps' related information embedded in the HTTP URI as it is a legacy HTTP client sending the request. Use case #2 is also expected to be similar. In contrast, in use case #3, it is likely that the HTTP client will specifically embed 'coap' or 'coaps' related information in the HTTP URI of the HTTP request to the HC Proxy.

5. URI Mapping

Though, in principle, a CoAP URI could be directly used by an HTTP client to de-reference a CoAP resource through an HC Proxy, the reality is that all major web browsers, networking libraries and command line tools do not allow making HTTP requests using URIs with a scheme 'coap' or 'coaps'.

Thus, there is a need for web applications to embed or "pack" a CoAP URI into an HTTP URI so that it can be (non-destructively) transported from the HTTP client to the HC Proxy. The HC Proxy can then "unpack" the CoAP URI and finally de-reference it via a CoAP request to the target Server.

URI Mapping is the term used in this document to describe the process through which the URI of a CoAP resource is transformed into an HTTP URI so that:

- o The requesting HTTP client can handle it;
- o The receiving HC Proxy can extract the intended CoAP URI unambiguously.

To this end, the remainder of this section will identify:

- o The default mechanism to map a CoAP URI into an HTTP URI;
- o The URI template format to express a class of CoAP-HTTP URI mapping functions;
- o The discovery mechanism based on CoRE Link Format [RFC6690] through which clients of an HC Proxy can dynamically learn about

the supported URI Mapping Template(s), as well as the URI where the HC Proxy function is anchored.

5.1. URI Terminology

In the remainder of this section, the following terms will be used with a distinctive meaning:

HC Proxy URI:

URI which refers to the HC Proxy function. It conforms to syntax defined in Section 2.7 of [RFC7230].

Target CoAP URI:

URI which refers to the (final) CoAP resource that has to be de-referenced. It conforms to syntax defined in Section 6 of [RFC7252]. Specifically, its scheme is either 'coap' or 'coaps'.

Hosting HTTP URI:

URI that conforms to syntax in Section 2.7 of [RFC7230]. Its authority component refers to an HC Proxy, whereas path and/or query component(s) embed the information used by an HC Proxy to extract the Target CoAP URI.

5.2. Null Mapping

The null mapping is the case where there is no Target CoAP URI appended to the HC Proxy URI. In other words, it is a "pure" HTTP URI that is sent to the HC Proxy. This would typically occur in situations like Use Case #1 described in Section 4, and the Proxy would typically be a Reverse Proxy. In this scenario, the HC Proxy will determine through its own private algorithms what the Target CoAP URI should be.

5.3. Default Mapping

The default mapping is for the Target CoAP URI to be appended as-is (with the only caveat discussed in Section 5.3.2) to the HC Proxy URI, to form the Hosting HTTP URI. This is the Effective Request URI (see Section 5.5 of [RFC7230]) that will then be sent by the HTTP client in the HTTP request to the HC Proxy.

For example: given an HC Proxy URI `https://p.example.com/hc/` and a Target CoAP URI `coap://s.example.com/light`, the resulting Hosting HTTP URI would be `https://p.example.com/hc/coap://s.example.com/light`.

Provided a correct Target CoAP URI, the Hosting HTTP URI resulting from the default mapping will be a syntactically valid HTTP URI. Furthermore, the Target CoAP URI can always be extracted unambiguously from the Hosting HTTP URI.

There is no default for the HC Proxy URI. Therefore, it is either known in advance, e.g., as a configuration preset, or dynamically discovered using the mechanism described in Section 5.5.

The default URI mapping function SHOULD be implemented and SHOULD be activated by default in an HC Proxy, unless there are valid reasons (e.g., application specific) to use a different mapping function.

5.3.1. Optional Scheme Omission

When constructing a Hosting HTTP URI by embedding a Target CoAP URI, the scheme (i.e., 'coap' or 'coaps'), the scheme component delimiter (":"), and the double slash ("/") preceding the authority MAY be omitted if a local default – not defined by this document – applies. If no prior mutual agreement exists between the client and the HC Proxy, then a Target CoAP URI without the scheme component is syntactically incorrect, and therefore:

- o It MUST NOT be emitted by clients;
- o It MUST elicit a suitable client error status (i.e., 4xx) by the HC Proxy.

5.3.2. Encoding Caveats

When the authority of the Target CoAP URI is given as an IPv6address, then the surrounding square brackets must be percent-encoded in the Hosting HTTP URI, in order to comply with the syntax defined in Section 3.3. of [RFC3986] for a URI path segment. E.g.:
coap://[2001:db8::1]/light?on becomes
https://p.example.com/hc/coap://%5B2001:db8::1%5D/light?on. (Note that the percent-encoded square brackets shall be reverted to their non-percent-encoded form when the HC Proxy unpacks the Target CoAP URI.)

Everything else can be safely copied verbatim from the Target CoAP URI to the Hosting HTTP URI.

5.4. URI Mapping Template

This section defines a format for the URI template [RFC6570] used by an HC Proxy to inform its clients about the expected syntax for the Hosting HTTP URI. This will then be used by the HTTP client to

construct the Effective Request URI to be sent in the HTTP request to the HC Proxy.

When instantiated, a URI Mapping Template is always concatenated to an HC Proxy URI provided by the HC Proxy via discovery (see Section 5.5), or by other means.

A simple form (Section 5.4.1) and an enhanced form (Section 5.4.2) are provided to fit different users' requirements.

Both forms are expressed as level 2 URI templates [RFC6570] to take care of the expansion of values that are allowed to include reserved URI characters. The syntax of all URI formats is specified in this section in Augmented Backus-Naur Form (ABNF) [RFC5234].

5.4.1. Simple Form

The simple form MUST be used for mappings where the Target CoAP URI is going to be copied (using rules of Section 5.3.2) at some fixed position into the Hosting HTTP URI.

The "tu" template variable is intended to be used in a template definition to represent a Target CoAP URI:

```
tu = [ ( "coap:" / "coaps:" ) "://" ] host [ ":" port ] path-abempty
      [ "?" query ]
```

Note that the same considerations as in Section 5.3.1 apply, in that the CoAP scheme may be omitted from the Hosting HTTP URI.

5.4.1.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use `https://p.example.com/hc/` as the HC Proxy URI. Note that these examples all define mapping templates that deviate from the default template of Section 5.3 in order to illustrate the use of the above template variables.

1. Target CoAP URI is a query argument of the Hosting HTTP URI:

```
?target_uri={+tu}
coap://s.example.com/light
=> https://p.example.com/hc/?target_uri=coap://s.example.com/light
whereas
coaps://s.example.com/light
=> https://p.example.com/hc/?target_uri=coaps://s.example.com/light
```

2. Target CoAP URI in the path component of the Hosting HTTP URI:

```
forward/{+tu}
coap://s.example.com/light
=> https://p.example.com/hc/forward/coap://s.example.com/light
whereas
coaps://s.example.com/light
=> https://p.example.com/hc/forward/coaps://s.example.com/light
```

3. Target CoAP URI is a query argument of the Hosting HTTP URI;
client decides to omit the scheme because a default is agreed
beforehand between client and proxy:

```
?coap_uri={+tu}
coap://s.example.com/light
=> https://p.example.com/hc/?coap_uri=s.example.com/light
```

5.4.2. Enhanced Form

The enhanced form can be used to express more sophisticated mappings of the Target CoAP URI into the Hosting HTTP URI, i.e., mappings that do not fit into the simple form.

There MUST be at most one instance of each of the following template variables in a template definition:

```
s  = "coap" / "coaps" ; from [RFC7252], Sections 6.1 and 6.2
hp = host [":" port]   ; from [RFC3986], Sections 3.2.2 and 3.2.3
p  = path-abempty      ; from [RFC3986], Section 3.3
q  = query              ; from [RFC3986], Section 3.4
qq = [ "?" query ]     ; qq is empty if and only if 'query' is empty
```

The qq form is used when the path and the (optional) query components are to be copied verbatim from the Target CoAP URI into the Hosting HTTP URI, i.e., as "{+p}{+qq}". Instead, the q form is used when the query and path are mapped as separate entities, e.g., as in "coap_path={+p}&coap_query={+q}".

5.4.2.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use `https://p.example.com/hc/` as the HC Proxy URI.

1. Target CoAP URI components in path segments, and optional query in query component:

```
{+s}/{+hp}{+p}{+qq}

coap://s.example.com/light

=> https://p.example.com/hc/coap/s.example.com/light

whereas

coap://s.example.com/light?on

=> https://p.example.com/hc/coap/s.example.com/light?on
```

2. Target CoAP URI components split in individual query arguments:

```
?s={+s}&hp={+hp}&p={+p}&q={+q}

coap://s.example.com/light

=> https://p.example.com/hc/?s=coap&hp=s.example.com&p=/light&q=

whereas

coaps://s.example.com/light?on

=> https://p.example.com/hc/?s=coaps&hp=s.example.com&p=/light&q=on
```

5.5. Discovery

In order to accommodate site-specific needs while allowing third parties to discover the proxy function, the HC Proxy SHOULD publish information related to the location and syntax of the HC Proxy function using the CoRE Link Format [RFC6690] interface.

To this aim a new Resource Type, "core.hc", is defined in this document. It can be used as the value for the "rt" attribute in a query to the /.well-known/core in order to locate the URI where the HC Proxy function is anchored, i.e., the HC Proxy URI.

Along with it, the new target attribute "hct" is defined in this document. This attribute MAY be returned in a "core.hc" link to provide the URI Mapping Template associated with the mapping resource. The default template given in Section 5.3, i.e., {+tu}, MUST be assumed if no "hct" attribute is found in the returned link. If a "hct" attribute is present in the returned link, then a client MUST use it to create the Hosting HTTP URI.

The URI mapping SHOULD be discoverable (as specified in [RFC6690]) on both the HTTP and the CoAP side of the HC Proxy, with one important difference: on the CoAP side the link associated with the "core.hc" resource needs an explicit anchor referring to the HTTP origin [RFC6454], while on the HTTP interface the link context is already the HTTP origin carried in the request's Host header, and doesn't have to be made explicit.

5.5.1. Examples

- o The first example exercises the CoAP interface and assumes that the default template, {+tu}, is used. For example, a smartphone may discover the public HC Proxy before leaving the home network. Then when outside the home network, the smartphone will be able to query the appropriate home sensor.


```
Req:  GET coap://[ff02::1]/.well-known/core?rt=core.hc
Res:  2.05 Content
      </hc/>;anchor="https://p.example.com";rt="core.hc"
```

- o The second example - also on the CoAP side of the HC Proxy - uses a custom template, i.e., one where the CoAP URI is carried inside the query component, thus the returned link carries the URI template to be used in an explicit "hct" attribute:

```
Req:  GET coap://[ff02::1]/.well-known/core?rt=core.hc
Res:  2.05 Content
      </hc/>;anchor="https://p.example.com";
      rt="core.hc";hct="?uri={+tu}"
```

On the HTTP side, link information can be serialized in more than one way:

- o using the 'application/link-format' content type:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com
Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format
      Content-Length: 18
      </hc/>;rt="core.hc"
```

- o using the 'application/link-format+json' content type as defined in [I-D.ietf-core-links-json]:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com
Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format+json
      Content-Length: 31
      [{"href":"/hc/","rt":"core.hc"}]
```

- o using the Link header:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Link: </hc/>;rt="core.hc"
```

6. Media Type Mapping

6.1. Overview

An HC Proxy needs to translate HTTP media types (Section 3.1.1.1 of [RFC7231]) and content encodings (Section 3.1.2.2 of [RFC7231]) into CoAP content formats (Section 12.3 of [RFC7252]) and vice versa.

Media type translation can happen in GET, PUT or POST requests going from HTTP to CoAP, and in 2.xx (i.e., successful) responses going from CoAP to HTTP. Specifically, PUT and POST need to map both the Content-Type and Content-Encoding HTTP headers into a single CoAP Content-Format option, whereas GET needs to map Accept and Accept-Encoding HTTP headers into a single CoAP Accept option. To generate the HTTP response, the CoAP Content-Format option is mapped back to a suitable HTTP Content-Type and Content-Encoding combination.

An HTTP request carrying a Content-Type and Content-Encoding combination which the HC Proxy is unable to map to an equivalent CoAP Content-Format, SHALL elicit a 415 (Unsupported Media Type) response by the HC Proxy.

On the content negotiation side, failure to map Accept and Accept-* headers SHOULD be silently ignored: the HC Proxy SHOULD therefore forward as a CoAP request with no Accept option. The HC Proxy thus disregards the Accept/Accept-* header fields by treating the response as if it is not subject to content negotiation, as mentioned in Sections 5.3.* of [RFC7231]. However, an HC Proxy implementation is free to attempt mapping a single Accept header in a GET request to multiple CoAP GET requests, each with a single Accept option, which are then tried in sequence until one succeeds. Note that an HTTP Accept /* MUST be mapped to a CoAP request without Accept option.

While the CoAP to HTTP direction has always a well-defined mapping (with the exception examined in Section 6.2), the HTTP to CoAP direction is more problematic because the source set, i.e., potentially 1000+ IANA registered media types, is much bigger than

the destination set, i.e., the mere 6 values initially defined in Section 12.3 of [RFC7252].

Depending on the tight/loose coupling with the application(s) for which it proxies, the HC Proxy could implement different media type mappings.

When tightly coupled, the HC Proxy knows exactly which content formats are supported by the applications, and can be strict when enforcing its forwarding policies in general, and the media type mapping in particular.

On the other hand, when the HC Proxy is a general purpose ALG, being too strict could significantly reduce the amount of traffic that it would be able to successfully forward. In this case, the "loose" media type mapping detailed in Section 6.3 MAY be implemented.

The latter grants more evolution of the surrounding ecosystem, at the cost of allowing more attack surface. In fact, as a result of such strategy, payloads would be forwarded more liberally across the unconstrained/constrained network boundary of the communication path.

6.2. 'application/coap-payload' Media Type

If the HC Proxy receives a CoAP response with a Content-Format that it does not recognize (e.g., because the value has been registered after the proxy has been deployed, or the CoAP server uses an experimental value which is not registered), then the HC Proxy SHALL return a generic "application/coap-payload" media type with numeric parameter "cf" as defined in Section 9.2.

For example, the CoAP content format '60' ("application/cbor") would be represented by "application/coap-payload;cf=60", if the HC Proxy doesn't recognize the content format '60'.

A HTTP client may use the media type "application/coap-payload" as a means to send a specific content format to a CoAP server via an HC Proxy if the client has determined that the HC Proxy does not directly support the type mapping it needs. This case may happen when dealing for example with newly registered, yet to be registered, or experimental CoAP content formats. However, unless explicitly configured to allow pass-through of unknown content formats, the HC Proxy SHOULD NOT forward requests carrying a Content-Type or Accept header with an "application/coap-payload", and return an appropriate client error instead.

6.3. Loose Media Type Mapping

By structuring the type information in a super-class (e.g., "text") followed by a finer grained sub-class (e.g., "html"), and optional parameters (e.g., "charset=utf-8"), Internet media types provide a rich and scalable framework for encoding the type of any given entity.

This approach is not applicable to CoAP, where Content Formats conflate an Internet media type (potentially with specific parameters) and a content encoding into one small integer value.

To remedy this loss of flexibility, we introduce the concept of a "loose" media type mapping, where media types that are specializations of a more generic media type can be aliased to their super-class and then mapped (if possible) to one of the CoAP content formats. For example, "application/soap+xml" can be aliased to "application/xml", which has a known conversion to CoAP. In the context of this "loose" media type mapping, "application/octet-stream" can be used as a fallback when no better alias is found for a specific media type.

Table 1 defines the default lookup table for the "loose" media type mapping. It is expected that an implementation can refine it either given application-specific knowledge, or because new Content-Formats are defined. Given an input media type, the table returns its best generalized media type using the most specific match i.e., the table entries are compared to the input in top to bottom order until an entry matches.

Internet media type pattern	Generalized media type
application/*+xml	application/xml
application/*+json	application/json
application/*+cbor	application/cbor
text/xml	application/xml
text/*	text/plain
/	application/octet-stream

Table 1: Media type generalization lookup table

The "loose" media type mapping is an OPTIONAL feature. Implementations supporting this kind of mapping should provide a flexible way to define the set of media type generalizations allowed.

6.4. Media Type to Content Format Mapping Algorithm

This section defines the algorithm used to map an HTTP Internet media type to its correspondent CoAP content format; it can be used as a building block for translating HTTP Content-Type and Accept headers into CoAP Content-Format and Accept Options.

The algorithm uses an IANA-maintained table, "CoAP Content-Formats", as established by Section 12.3 of [RFC7252] plus, possibly, any locally defined extension of it. Optionally, the table and lookup mechanism described in Section 6.3 can be used if the implementation chooses so.

Note that the algorithm assumes an "identity" Content-Encoding and expects the resource body has been already successfully content-decoded or transcoded to the desired format.

In the following (Figure 2):

- o `media_type` is the media type to translate;
- o `coap_cf_registry` is a lookup table matching the CoAP Content Format Registry;
- o `loose_mapper` is an optional lookup table describing the loose media type mappings (e.g., the one defined in Table 1);

The full source code is provided in Appendix A.

```
def mt2cf(media_type, encoding=None,
          coap_cf_registry=CoAPContentFormatRegistry(),
          loose_mapper=None):
    """Return a CoAP Content-Format given an Internet Media Type and
    its optional encoding. The current (as of 2016/10/24) CoAP
    Content Format Registry is supplied by default. An optional
    'loose-mapping' implementation can be supplied by the caller."""
    assert media_type is not None
    assert coap_cf_registry is not None

    # Lookup the CoAP Content-Formats registry
    content_format = coap_cf_registry.lookup(media_type, encoding)

    # If an exact match is not found and a loose mapper has been
    # supplied, try to use it to get a media type with which to
    # re-try the CoAP Content-Formats registry lookup.
    if content_format is None and loose_mapper is not None:
        content_format = coap_cf_registry.lookup(
            loose_mapper.lookup(media_type), encoding)

    return content_format
```

Figure 2

6.5. Content Transcoding

6.5.1. General

Payload content transcoding is an OPTIONAL feature. Implementations supporting this feature should provide a flexible way to define the set of transcodings allowed.

The HC Proxy might decide to transcode the received representation to a different (compatible) format when an optimized version of a specific format exists. For example, a XML-encoded resource could be transcoded to Efficient XML Interchange (EXI) format, or a JSON-encoded resource into CBOR [RFC7049], effectively achieving compression without losing any information.

However, there are a few important factors to keep in mind when enabling a transcoding function:

1. Maliciously crafted inputs coming from the HTTP side might inflate in size (see for example Section 4.2 of [RFC7049]), therefore creating a security threat for both the HC Proxy and the target resource;

2. Transcoding can lose information in non-obvious ways. For example, encoding a XML document using schema-informed EXI encoding leads to a loss of information when the destination does not know the exact schema version used by the encoder. That means that whenever the HC Proxy transcodes an application/XML to application/EXI in-band metadata could be lost.
3. When content-type is mapped, there is a risk that the content with the destination type would have malware not active in the source type.

It is crucial that these risks are well understood and carefully weighed against the actual benefits before deploying the transcoding function.

6.5.2. CoRE Link Format

The CoRE Link Format [RFC6690] is a set of links (i.e., URIs and their formal relationships) which is carried as content payload in a CoAP response. These links usually include CoAP URIs that might be translated by the HC Proxy to the correspondent HTTP URIs using the implemented URI mapping function (see Section 5). Such a process would inspect the forwarded traffic and attempt to re-write the body of resources with an application/link-format media type, mapping the embedded CoAP URIs to their HTTP counterparts. Some potential issues with this approach are:

1. The client may be interested in retrieving original (unaltered) CoAP payloads through the HC Proxy, not modified versions.
2. Tampering with payloads is incompatible with resources that are integrity protected (although this is a problem with transcoding in general).
3. The HC Proxy needs to fully understand [RFC6690] syntax and semantics, otherwise there is an inherent risk to corrupt the payloads.

Therefore, CoRE Link Format payload should only be transcoded at the risk and discretion of the proxy implementer.

6.5.3. Diagnostic Messages

CoAP responses may, in certain error cases, contain a diagnostic message in the payload explaining the error situation, as described in Section 5.5.2 of [RFC7252]. If present, the CoAP response diagnostic payload SHOULD be copied in the HTTP response body. The CoAP diagnostic message MUST NOT be copied into the HTTP reason-

phrase, since it potentially contains CR-LF characters which are incompatible with HTTP reason-phrase syntax.

7. Response Code Mapping

Table 2 defines the HTTP response status codes to which each CoAP response code SHOULD be mapped. Multiple appearances of a HTTP status code in the second column indicates multiple equivalent HTTP responses are possible based on the same CoAP response code, depending on the conditions cited in the Notes (third column and text below table).

CoAP Response Code	HTTP Status Code	Note
2.01 Created	201 Created	1
2.02 Deleted	200 OK	2
	204 No Content	2
2.03 Valid	304 Not Modified	3
	200 OK	4
2.04 Changed	200 OK	2
	204 No Content	2
2.05 Content	200 OK	
2.31 Continue	N/A	10
4.00 Bad Request	400 Bad Request	
4.01 Unauthorized	403 Forbidden	5
4.02 Bad Option	400 Bad Request	6
4.02 Bad Option	500 Internal Server Error	6
4.03 Forbidden	403 Forbidden	
4.04 Not Found	404 Not Found	
4.05 Method Not Allowed	400 Bad Request	7
4.06 Not Acceptable	406 Not Acceptable	
4.08 Request Entity Incomplt.	N/A	10
4.12 Precondition Failed	412 Precondition Failed	
4.13 Request Ent. Too Large	413 Payload Too Large	11
4.15 Unsupported Content-Fmt.	415 Unsupported Media Type	
5.00 Internal Server Error	500 Internal Server Error	
5.01 Not Implemented	501 Not Implemented	
5.02 Bad Gateway	502 Bad Gateway	
5.03 Service Unavailable	503 Service Unavailable	8
5.04 Gateway Timeout	504 Gateway Timeout	
5.05 Proxying Not Supported	502 Bad Gateway	9

Table 2: CoAP-HTTP Response Code Mappings

Notes:

1. A CoAP server may return an arbitrary format payload along with this response. If present, this payload MUST be returned as entity in the HTTP 201 response. Section 7.3.2 of [RFC7231] does not put any requirement on the format of the entity. (In the past, [RFC2616] did.)
2. The HTTP code is 200 or 204 respectively for the case that a CoAP server returns a payload or not. [RFC7231] Section 5.3 requires code 200 in case a representation of the action result is returned for DELETE/POST/PUT, and code 204 if not. Hence, a proxy MUST transfer any CoAP payload contained in a CoAP 2.02 response to the HTTP client using a 200 OK response.
3. HTTP code 304 (Not Modified) is sent if the HTTP client performed a conditional HTTP request and the CoAP server responded with 2.03 (Valid) to the corresponding CoAP validation request. Note that Section 4.1 of [RFC7232] puts some requirements on header fields that must be present in the HTTP 304 response.
4. A 200 response to a CoAP 2.03 occurs only when the HC Proxy, for efficiency reasons, is running a local cache. An unconditional HTTP GET which produces a cache-hit, could trigger a re-validation (i.e., a conditional GET) on the CoAP side. The proxy receiving 2.03 updates the freshness of its cached representation and returns it to the HTTP client.
5. A HTTP 401 Unauthorized (Section 3.1 of [RFC7235]) response is not applicable because there is no equivalent in CoAP of WWW-Authenticate which is mandatory in a HTTP 401 response.
6. If the proxy has a way to determine that the Bad Option is due to the straightforward mapping of a client request header into a CoAP option, then returning HTTP 400 (Bad Request) is appropriate. In all other cases, the proxy MUST return HTTP 500 (Internal Server Error) stating its inability to provide a suitable translation to the client's request.
7. A CoAP 4.05 (Method Not Allowed) response SHOULD normally be mapped to a HTTP 400 (Bad Request) code, because the HTTP 405 response would require specifying the supported methods - which are generally unknown. In this case the HC Proxy SHOULD also return a HTTP reason-phrase in the HTTP status line that starts with the string "CoAP server returned 4.05" in order to facilitate troubleshooting. However, if the HC Proxy has more granular information about the supported methods for the requested resource (e.g., via a Resource Directory ([I-D.ietf-core-resource-directory])) then it MAY send back a

HTTP 405 (Method Not Allowed) with a properly filled in "Allow" response-header field (Section 7.4.1 of [RFC7231]).

8. The value of the HTTP "Retry-After" response-header field is taken from the value of the CoAP Max-Age Option, if present.
9. This CoAP response can only happen if the proxy itself is configured to use a CoAP forward-proxy (Section 5.7 of [RFC7252]) to execute some, or all, of its CoAP requests.
10. Only used in CoAP blockwise transfer [RFC7959] between HC Proxy and CoAP server; never translated into a HTTP response.
11. Only returned to the HTTP client if the HC Proxy was unable to successfully complete the request by retrying it with CoAP blockwise transfer; see Section 8.3.

8. Additional Mapping Guidelines

8.1. Caching and Congestion Control

An HC Proxy should cache CoAP responses, and reply whenever applicable with a cached representation of the requested resource.

If the HTTP client drops the connection after the HTTP request was made, an HC Proxy should wait for the associated CoAP response and cache it if possible. Subsequent requests to the HC Proxy for the same resource can use the result present in cache, or, if a response has still to come, the HTTP requests will wait on the open CoAP request.

According to [RFC7252], a proxy must limit the number of outstanding requests to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the HC Proxy should also put a limit on the number of concurrent CoAP requests pending on the same constrained network; further incoming requests may either be queued or dropped (returning 503 Service Unavailable). This limit and the proxy queueing/dropping behavior should be configurable.

Highly volatile resources that are being frequently requested may be observed [RFC7641] by the HC Proxy to keep their cached representation fresh while minimizing the amount of CoAP traffic in the constrained network (see Section 8.2).

8.2. Cache Refresh via Observe

There are cases where using the CoAP observe protocol [RFC7641] to handle proxy cache refresh is preferable to the validation mechanism based on ETag as defined in [RFC7252]. Such scenarios include sleepy CoAP nodes - with possibly high variance in requests' distribution - which would greatly benefit from a server-driven cache update mechanism. Ideal candidates for CoAP observe are also crowded or very low throughput networks, where reduction of the total number of exchanged messages is an important requirement.

This subsection aims at providing a practical evaluation method to decide whether refreshing a cached resource R is more efficiently handled via ETag validation or by establishing an observation on R. The idea being that the HC Proxy proactively installs an observation on a "popular enough" resource and actively monitors:

- a. Its update pattern on the CoAP side; and
- b. The request pattern on the HTTP side;

and uses the formula below to determine whether the observation should be kept alive or shut down.

Let T_R be the mean time between two client requests to resource R, let T_C be the mean time between two representation changes of R, and let M_R be the mean number of CoAP messages per second exchanged to and from resource R. If we assume that the initial cost for establishing the observation is negligible, an observation on R reduces M_R if and only if $T_R < 2 * T_C$ with respect to using ETag validation, that is if and only if the mean arrival rate of requests for resource R is greater than half the change rate of R.

When observing the resource R, M_R is always upper bounded by $2/T_C$.

8.3. Use of CoAP Blockwise Transfer

An HC Proxy SHOULD support CoAP blockwise transfers [RFC7959] to allow transport of large CoAP payloads while avoiding excessive link-layer fragmentation in constrained networks, and to cope with small datagram buffers in CoAP endpoints as described in [RFC7252] Section 4.6.

An HC Proxy SHOULD attempt to retry a payload-carrying CoAP PUT or POST request with blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. An HC Proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than

BLOCKWISE_THRESHOLD bytes. The value of BLOCKWISE_THRESHOLD is implementation-specific; for example, it can be:

- o Calculated based on a known or typical UDP datagram buffer size for CoAP endpoints, or
- o Set to N times the known size of a link-layer frame in a constrained network where e.g., N=5, or
- o Preset to a known IP MTU value, or
- o Set to a known Path MTU value.

The value BLOCKWISE_THRESHOLD, or the parameters from which it is calculated, should be configurable in a proxy implementation. The maximum block size the proxy will attempt to use in CoAP requests should also be configurable.

The HC Proxy SHOULD detect CoAP endpoints not supporting blockwise transfers. This can be done by checking for a 4.02 (Bad Option) response returned by an endpoint in response to a CoAP request with a Block* Option, and subsequent absence of the 4.02 in response to the same request without Block* Options. This allows the HC Proxy to be more efficient, not attempting repeated blockwise transfers to CoAP servers that do not support it.

8.4. CoAP Multicast

An HC Proxy MAY support CoAP multicast. If it does, the HC Proxy sends out a multicast CoAP request if the Target CoAP URI's authority is a multicast IP literal or resolves to a multicast IP address. If the HC Proxy does not support CoAP multicast, it SHOULD respond 403 (Forbidden) to any valid HTTP request that maps to a CoAP multicast request.

Details related to supporting CoAP multicast are currently out of scope of this document since in a proxy scenario an HTTP client typically expects to receive a single response, not multiple. However, an HC Proxy that implements CoAP multicast may include application-specific functions to aggregate multiple CoAP responses into a single HTTP response. We suggest using the "application/http" internet media type (Section 8.3.2 of [RFC7230]) to enclose a set of one or more HTTP response messages, each representing the mapping of one CoAP response.

For further considerations related to the handling of multicast requests, see Section 10.1.

8.5. Timeouts

If the CoAP server takes a long time in responding, the HTTP client or any other proxy in between may timeout. Further discussion of timeouts in HTTP is available in Section 6.2.4 of [RFC7230].

An HC Proxy MUST define an internal timeout for each pending CoAP request, because the CoAP server may silently die before completing the request. Assuming the Proxy uses confirmable CoAP requests, such timeout value T SHOULD be at least

$$T = \text{MAX_RTT} + \text{MAX_SERVER_RESPONSE_DELAY}$$

where MAX_RTT is defined in [RFC7252] and MAX_SERVER_RESPONSE_DELAY is defined in [RFC7390].

9. IANA Considerations

9.1. New 'core.hc' Resource Type

This document registers a new Resource Type (rt=) Link Target Attribute, 'core.hc', in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

Attribute Value: core.hc

Description: HTTP to CoAP mapping base resource.

Reference: See Section 5.5.

9.2. New 'coap-payload' Internet Media Type

This document defines the "application/coap-payload" media type with a single parameter "cf". This media type represents any payload that a CoAP message can carry, having a content format that can be identified by an integer in range 0-65535 corresponding to a CoAP Content-Format parameter ([RFC7252], Section 12.3). The parameter "cf" is the integer defining the CoAP content format.

Type name: application

Subtype name: coap-payload

Required parameters: cf (CoAP Content-Format integer in range 0-65535 denoting the content format of the CoAP payload carried, as defined by the "CoAP Content-Formats" subregistry that is part of the "Constrained RESTful Environments (CoRE) Parameters" registry.)

Optional parameters: None

Encoding considerations: Common use is BINARY. The specific CoAP content format encoding considerations for the selected Content-Format (cf parameter) apply. The encoding can vary based on the value of the cf parameter.

Security considerations: The specific CoAP content format security considerations for the selected Content-Format (cf parameter) apply.

Interoperability considerations: This media type can never be used directly in CoAP messages because there are no means available to encode the mandatory 'cf' parameter in CoAP.

Published specification: (this I-D - TBD)

Applications that use this media type: HTTP-to-CoAP Proxies.

Fragment identifier considerations: CoAP does not support URI fragments; therefore a CoAP payload fragment cannot be identified. Fragments are not applicable for this media type.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information:

Esko Dijk ("esko@ieee.org")

Intended usage: COMMON

Restrictions on usage:

An application (or user) can only use this media type if it has to represent a CoAP payload of which the specified CoAP Content-Format is an unrecognized number; such that a proper translation directly to the equivalent HTTP media type is not possible.

Author: CoRE WG

Change controller: IETF

Provisional registration: No

10. Security Considerations

The security considerations in Section 9.2 of [RFC7230] apply in full to the HC Proxy. This section discusses security aspects and requirements that are specific to the deployment and operation of an HC Proxy.

An HC Proxy located at the boundary of a constrained network is an easy single point of failure for reducing availability. As such, special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it has fewer limitations than the constrained devices it is serving. In particular, its quality of implementation and operation - i.e., use of current software development practices, careful selection of third party libraries, sane configuration defaults, an expedited way to upgrade a running instance - are all essential attributes of the HC Proxy.

The correctness of request parsing in general (including any content transcoding), and of URI translation in particular, is essential to the security of the HC Proxy function. This is especially true when the internal network hosts devices with genuinely limited capabilities. For this purpose, see also Sections 9.3, 9.4, 9.5 and 9.6 of [RFC7230] for well-known issues related to HTTP request parsing and Section 11.1 of [RFC7252] for an overview of CoAP specific concerns related to URI processing - in particular, the potential impact on access control mechanisms that are based on URIs.

An HC Proxy MUST implement TLS with PSK [RFC4279] and SHOULD implement TLS [RFC5246] with support for client authentication using X.509 certificates. A prerequisite of the latter is the availability of a Certification Authority (CA) to issue suitable certificates. Although this can be a challenging requirement in certain application scenarios, it is worth noting that there exist open-source tools (e.g., [OpenSSL]) that can be used to set up and operate an application-specific CA.

By default, the HC Proxy MUST authenticate all incoming requests prior to forwarding them to the CoAP server. This default behavior MAY be explicitly disabled by an administrator.

The following subparagraphs categorize and discuss a set of specific security issues related to the translation, caching and forwarding functionality exposed by an HC Proxy.

10.1. Multicast

Multicast requests impose a non-trivial cost on the constrained network and endpoints and might be exploited as a DoS attack vector (see also Section 10.2). From a privacy perspective, they can be used to gather detailed information about the resources hosted in the constrained network. For example, an outsider that is able to successfully query the /.well-known/core could obtain a comprehensive list of the target's home appliances and devices. From a security perspective, they can be used to carry out a network reconnaissance attack to gather information about possible vulnerabilities that could be exploited at a later point in time. For these reasons, it is RECOMMENDED that requests to multicast resources are access controlled with a default-deny policy. It is RECOMMENDED that the requestor of a multicast resource be strongly authenticated. If privacy and / or security are first class requirements, for example whenever the HTTP request transits through the public Internet, the request SHOULD be transported over a mutually authenticated and encrypted TLS connection.

10.2. Traffic Overflow

Due to the typically constrained nature of CoAP nodes, particular attention should be given to the implementation of traffic reduction mechanisms (see Section 8.1), because an inefficient proxy implementations can be targeted by unconstrained Internet attackers. Bandwidth or complexity involved in such attacks is very low.

An amplification attack to the constrained network may be triggered by a multicast request generated by a single HTTP request which is mapped to a CoAP multicast resource, as discussed in Section 11.3 of [RFC7252].

The risk likelihood of this amplification technique is higher than an amplification attack carried out by a malicious constrained device (e.g., ICMPv6 flooding, like Packet Too Big, or Parameter Problem on a multicast destination [RFC4732]) since it does not require direct access to the constrained network.

The feasibility of this attack, which disrupts availability of the targeted CoAP server, can be limited by access controlling the exposed multicast resources, so that only known/authorized users can access such URIs.

10.3. Handling Secured Exchanges

An HTTP request can be sent to the HC Proxy over a secured connection. However, there may not always exist a secure connection mapping to CoAP. For example, a secure distribution method for multicast traffic is complex and may not be implemented (see [RFC7390]).

An HC Proxy should implement rules for security context translations. For example, all 'https' unicast requests are translated to 'coaps' requests, or 'https' requests are translated to unsecured 'coap' requests. Another rule could specify the security policy and parameters used for DTLS sessions [RFC7925]. Such rules will largely depend on the application and network context in which the HC Proxy operates. These rules should be configurable.

It is RECOMMENDED that, by default, accessing a 'coaps' URI is only allowed from a corresponding 'https' URI.

By default, an HC Proxy SHOULD reject any secured CoAP client request (i.e., one with a 'coaps' scheme) if there is no configured security policy mapping. This recommendation may be relaxed in case the destination network is believed to be secured by other means. Assuming that CoAP nodes are isolated behind a firewall as in the HC Proxy deployment shown in Figure 1, the HC Proxy may be configured to translate the incoming HTTPS request using plain CoAP (NoSec mode).

10.4. URI Mapping

The following risks related to the URI mapping described in Section 5 and its use by HC Proxy have been identified:

DoS attack on the constrained/CoAP network.

Mitigation: by default deny any Target CoAP URI whose authority is (or maps to) a multicast address. Then explicitly white-list multicast resources/authorities that are allowed to be de-referenced. See also Section 8.4.

Leaking information on the constrained/CoAP network resources and topology.

Mitigation: by default deny any Target CoAP URI (especially /.well-known/core is a resource to be protected), and then explicitly white-list resources that are allowed to be seen from outside.

The internal CoAP Target resource is totally transparent from outside.

Mitigation: implement an HTTPS-only interface, which makes the Target CoAP URI totally opaque to a passive attacker.

11. Acknowledgments

An initial version of Table 2 in Section 7 has been provided in revision -05 of the CoRE CoAP I-D. Special thanks to Peter van der Stok for countless comments and discussions on this document that contributed to its current structure and text.

Thanks to Abhijan Bhattacharyya, Alexey Melnikov, Brian Frank, Carsten Bormann, Christian Amsuess, Christian Groves, Cullen Jennings, Dorothy Gellert, Francesco Corazza, Francis Dupont, Hannes Tschofenig, Jaime Jimenez, Kathleen Moriarty, Kepeng Li, Kerry Lynn, Klaus Hartke, Larry Masinter, Linyi Tian, Michele Rossi, Michele Zorzi, Nicola Bui, Peter Saint-Andre, Sean Leonard, Spencer Dawkins, Stephen Farrell, Suresh Krishnan, Zach Shelby for helpful comments and discussions that have shaped the document.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n.251557.

12. References

12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <<http://www.rfc-editor.org/info/rfc4279>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<http://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<http://www.rfc-editor.org/info/rfc7231>>.
- [RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <<http://www.rfc-editor.org/info/rfc7232>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <<http://www.rfc-editor.org/info/rfc7235>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<http://www.rfc-editor.org/info/rfc7959>>.

12.2. Informative References

[Fielding]

Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", PhD Dissertation, University of California, Irvine, ISBN 0-599-87118-0, 2000.

[I-D.ietf-core-links-json]

Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-06 (work in progress), July 2016.

[I-D.ietf-core-resource-directory]

Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-09 (work in progress), October 2016.

[OpenSSL]

The OpenSSL Project, , "ca - sample minimal CA application", 1998-2016,
<<https://www.openssl.org/docs/manmaster/man1/ca.html>>.

[RFC2616]

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999,
<<http://www.rfc-editor.org/info/rfc2616>>.

[RFC3040]

Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, DOI 10.17487/RFC3040, January 2001,
<<http://www.rfc-editor.org/info/rfc3040>>.

[RFC4732]

Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006,
<<http://www.rfc-editor.org/info/rfc4732>>.

[RFC6454]

Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011,
<<http://www.rfc-editor.org/info/rfc6454>>.

[RFC7049]

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.

- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<http://www.rfc-editor.org/info/rfc7390>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<http://www.rfc-editor.org/info/rfc7925>>.
- [W3C.REC-html5-20141028] Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, E., and S. Pfeiffer, "HTML5", W3C Recommendation REC-html5-20141028, 2014, <<http://www.w3.org/TR/2014/REC-html5-20141028>>.

Appendix A. Media Type Mapping Source Code

```
#!/usr/bin/env python

import unittest
import re

class CoAPContentFormatRegistry(object):
    """Map an Internet media type (and optional inherent encoding) to a
    CoAP content format.
    """
    TEXT_PLAIN = 0
    LINK_FORMAT = 40
    XML = 41
    OCTET_STREAM = 42
    EXI = 47
    JSON = 50
    CBOR = 60
    GROUP_JSON = 256

# http://www.iana.org/assignments/core-parameters/core-parameters.xhtml
# as of 2016/10/24.
LOOKUP_TABLE = {
    ("text/plain;charset=utf-8", None): TEXT_PLAIN,
```

```

        ("application/link-format", None): LINK_FORMAT,
        ("application/xml", None): XML,
        ("application/octet-stream", None): OCTET_STREAM,
        ("application/exi", None): EXI,
        ("application/json", None): JSON,
        ("application/cbor", None): CBOR,
        ("application/coap-group+json", "utf-8"): GROUP_JSON,
    }

    def lookup(self, media_type, encoding):
        """Return the CoAP Content Format matching the supplied
        media type (and optional encoding), or None if no
        match can be found."""
        return CoAPContentFormatRegistry.LOOKUP_TABLE.get(
            (media_type, encoding), None)

class LooseMediaTypeMapper(object):
    # Order matters in this table: more specific types have higher rank
    # compared to less specific types.
    # This code only performs a shallow validation of acceptable
    # characters, and assumes overall validation of media type and
    # subtype has been done beforehand.
    LOOKUP_TABLE = [
        (re.compile("application/.*\+xml$"), "application/xml"),
        (re.compile("application/.*\+json$"), "application/json"),
        (re.compile("application/.*\+cbor$"), "application/cbor"),
        (re.compile("text/xml$"), "application/xml"),
        (re.compile("text/[a-z\.\-\+]+$"), "text/plain; charset=utf-8"),
        (re.compile("[a-z]+/[a-z\.\-\+]+$"), "application/octet-stream")
    ]

    def lookup(self, media_type):
        """Return the best loose media type match available using
        the contents of LOOKUP_TABLE."""
        for entry in LooseMediaTypeMapper.LOOKUP_TABLE:
            if entry[0].match(media_type) is not None:
                return entry[1]
        return None

    def mt2cf(media_type, encoding=None,
              coap_cf_registry=CoAPContentFormatRegistry(),
              loose_mapper=None):
        """Return a CoAP Content-Format given an Internet Media Type and
        its optional encoding. The current (as of 2016/10/24) CoAP
        Content Format Registry is supplied by default. An optional
        'loose-mapping' implementation can be supplied by the caller."""

```

```
    assert media_type is not None
    assert coap_cf_registry is not None

    # Lookup the CoAP Content-Formats registry
    content_format = coap_cf_registry.lookup(media_type, encoding)

    # If an exact match is not found and a loose mapper has been
    # supplied, try to use it to get a media type with which to
    # re-try the CoAP Content-Formats registry lookup.
    if content_format is None and loose_mapper is not None:
        content_format = coap_cf_registry.lookup(
            loose_mapper.lookup(media_type), encoding)

    return content_format

class TestMT2CF(unittest.TestCase):

    def testMissingContentType(self):
        with self.assertRaises(AssertionError):
            mt2cf(None)

    def testMissingContentFormatRegistry(self):
        with self.assertRaises(AssertionError):
            mt2cf(None, coap_cf_registry=None)

    def testTextPlain(self):
        self.assertEqual(mt2cf("text/plain;charset=utf-8"),
            CoAPContentFormatRegistry.TEXT_PLAIN)

    def testLinkFormat(self):
        self.assertEqual(mt2cf("application/link-format"),
            CoAPContentFormatRegistry.LINK_FORMAT)

    def testXML(self):
        self.assertEqual(mt2cf("application/xml"),
            CoAPContentFormatRegistry.XML)

    def testOctetStream(self):
        self.assertEqual(mt2cf("application/octet-stream"),
            CoAPContentFormatRegistry.OCTET_STREAM)

    def testEXI(self):
        self.assertEqual(mt2cf("application/exi"),
            CoAPContentFormatRegistry.EXI)

    def testJSON(self):
        self.assertEqual(mt2cf("application/json"),
```

```
CoAPContentFormatRegistry.JSON)

def testCBOR(self):
    self.assertEqual(mt2cf("application/cbor"),
                     CoAPContentFormatRegistry.CBOR)

def testCoAPGroupJSON(self):
    self.assertEqual(mt2cf("application/coap-group+json",
                           "utf-8"),
                     CoAPContentFormatRegistry.GROUP_JSON)

def testUnknownMediaType(self):
    self.assertFalse(mt2cf("unknown/media-type"))

def testLooseXML1(self):
    self.assertEqual(
        mt2cf(
            "application/somesubtype+xml",
            loose_mapper=LooseMediaTypeMapper()),
        CoAPContentFormatRegistry.XML)

def testLooseXML2(self):
    self.assertEqual(
        mt2cf(
            "text/xml",
            loose_mapper=LooseMediaTypeMapper()),
        CoAPContentFormatRegistry.XML)

def testLooseJSON(self):
    self.assertEqual(
        mt2cf(
            "application/somesubtype+json",
            loose_mapper=LooseMediaTypeMapper()),
        CoAPContentFormatRegistry.JSON)

def testLooseCBOR(self):
    self.assertEqual(
        mt2cf(
            "application/somesubtype+cbor",
            loose_mapper=LooseMediaTypeMapper()),
        CoAPContentFormatRegistry.CBOR)

def testLooseText(self):
    self.assertEqual(
        mt2cf(
            "text/somesubtype",
            loose_mapper=LooseMediaTypeMapper()),
        CoAPContentFormatRegistry.TEXT_PLAIN)
```



```
def testLooseUnknown(self):
    self.assertEqual(
        mt2cf(
            "application/somesubtype-of-some-sort+format",
            loose_mapper=LooseMediaTypeMapper()),
        CoAPContentFormatRegistry.OCTET_STREAM)

def testLooseInvalidStartsWithNonAlpha(self):
    self.assertFalse(
        mt2cf(
            " application/somesubtype",
            loose_mapper=LooseMediaTypeMapper()))

def testLooseInvalidEndsWithUnexpectedChar(self):
    self.assertFalse(
        mt2cf(
            "application/somesubtype ",
            loose_mapper=LooseMediaTypeMapper()))

def testLooseInvalidUnexpectedCharInTheMiddle(self):
    self.assertFalse(
        mt2cf(
            "application /somesubtype",
            loose_mapper=LooseMediaTypeMapper()))

def testLooseInvalidNoSubType1(self):
    self.assertFalse(
        mt2cf(
            "application",
            loose_mapper=LooseMediaTypeMapper()))

def testLooseInvalidNoSubType2(self):
    self.assertFalse(
        mt2cf(
            "application/",
            loose_mapper=LooseMediaTypeMapper()))

if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Appendix B. Change Log

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-16 to ietf-17:

- o Intended status from Informational to Standards Track;

- o Stephen Farrell's DISCUSS
- o Added 2.31 and 4.08 CoAP response codes to the Response Code Mapping table.
- o Editorial fixes

Changes from ietf-15 to ietf-16 (Apps-Dir review):

- o Larry Masinter's comments.

Changes from ietf-14 to ietf-15 (IESG review):

- o Kathleen Moriarty's DISCUSS and COMMENT;
- o Stephen Farrell's COMMENT;
- o Suresh Krishnan DISCUSS;
- o Spencer Dawkins' DISCUSS and COMMENT;

Changes from ietf-13 to ietf-14:

- o Addressed Gen-ART and AD review comments.

Changes from ietf-12 to ietf-13 (Christian Amsuess' comments):

- o More missing slashes in URI mapping template examples.

Changes from ietf-11 to ietf-12 (2nd WGLC):

- o Addressed a few editorial issues (including a clarification on when to use qq vs q in the URI mapping template).
- o Fixed missing slash in one template example.
- o Added para about the need for future CoAP protocol elements to define their own HTTP mappings.

Changes from ietf-10 to ietf-11 (Chair review):

- o Removed cu/su distinction from the URI mapping template.
- o Addressed a few editorial issues.

Changes from ietf-09 to ietf-10:

- o Addressed Ticket #401 - Clarified that draft covers not only Reverse HC Proxy but that many parts also apply to Forward and Interception Proxies.
- o Clarified that draft concentrates on the HTTP-to-CoAP mapping direction (i.e., the HC Proxy is an HTTP server and a CoAP client).
- o Clarified the "null mapping" case where no CoAP URI information is embedded in the HTTP request URI.
- o Moved multicast related security text to the "Security Considerations" to consolidate all security information in one location.
- o Removed references to "placement" of proxy (e.g., server-side vs client-side) as is confusing and provides little added value.
- o Fixed version numbers on references that were corrupted in last revision due to outdated xml2rfc conversion tool local cache.
- o Various editorial improvements.

Changes from ietf-08 to ietf-09:

- o Clean up requirements language as per Klaus' comment.

Changes from ietf-07 to ietf-08:

- o Addressed WGLC review comments from Klaus Hartke as per the correspondence of March 9, 2016 on the CORE WG mailing list.

Changes from ietf-06 to ietf-07:

- o Addressed Ticket #384 - Section 5.4.1 describes briefly (informative) how to discover CoAP resources from an HTTP client.
- o Addressed Ticket #378 - For HTTP media type to CoAP content format mapping and vice versa: a new draft (TBD) may be proposed in CoRE which describes an approach for automatic updating of the media type mapping. This was noted in Section 6.1 but is otherwise outside the scope of this draft.
- o Addressed Ticket #377 - Added IANA section that defines a new HTTP media type "application/coap-payload" and created new Section 6.2 on how to use it.

- o Addressed Ticket #376 - Updated Table 2 (and corresponding note 7) to indicate that a CoAP 4.05 (Method Not Allowed) Response Code should be mapped to an HTTP 400 (Bad Request).
- o Added note to comply to ABNF when translating CoAP diagnostic payload to reason-phrase in Section 6.5.3.

Changes from ietf-05 to ietf-06:

- o Fully restructured the draft, bringing introductory text more to the front and allocating main sections to each of the key topics; addressing Ticket #379;
- o Addressed Ticket #382, fix of enhanced form URI template definition of q in Section 5.3.2;
- o Addressed Ticket #381, found a mapping 4.01 to 401 Unauthorized in Section 7;
- o Addressed Ticket #380 (Add IANA registration for "core.hc" Resource Type) in Section 9;
- o Addressed Ticket #376 (CoAP 4.05 response can't be translated to HTTP 405 by HC Proxy) in Section 7 by use of empty 'Allow' header;
- o Removed details on the pros and cons of HC Proxy placement options;
- o Addressed review comments of Carsten Bormann;
- o Clarified failure in mapping of HTTP Accept headers (Section 6.3);
- o Clarified detection of CoAP servers not supporting blockwise (Section 8.3);
- o Changed CoAP request timeout min value to MAX_RTT + MAX_SERVER_RESPONSE_DELAY (Section 8.6);
- o Added security section item (Section 10.3) related to use of CoAP blockwise transfers;
- o Many editorial improvements.

Changes from ietf-04 to ietf-05:

- o Addressed Ticket #366 (Mapping of CoRE Link Format payloads to be valid in HTTP Domain?) in Section 6.3.3.2 (Content Transcoding - CORE Link Format);

- o Addressed Ticket #375 (Add requirement on mapping of CoAP diagnostic payload) in Section 6.3.3.3 (Content Transcoding - Diagnostic Messages);
- o Addressed comment from Yusuke (<http://www.ietf.org/mail-archive/web/core/current/msg05491.html>) in Section 6.3.3.1 (Content Transcoding - General);
- o Various editorial improvements.

Changes from ietf-03 to ietf-04:

- o Expanded use case descriptions in Section 4;
- o Fixed/enhanced discovery examples in Section 5.4.1;
- o Addressed Ticket #365 (Add text on media type conversion by HTTP-CoAP proxy) in new Section 6.3.1 (Generalized media type mapping) and new Section 6.3.2 (Content translation);
- o Updated HTTPBis WG draft references to recently published RFC numbers.
- o Various editorial improvements.

Changes from ietf-02 to ietf-03:

- o Closed Ticket #351 "Add security implications of proposed default HTTP-CoAP URI mapping";
- o Closed Ticket #363 "Remove CoAP scheme in default HTTP-CoAP URI mapping";
- o Closed Ticket #364 "Add discovery of HTTP-CoAP mapping resource(s)".

Changes from ietf-01 to ietf-02:

- o Selection of single default URI mapping proposal as proposed to WG mailing list 2013-10-09.

Changes from ietf-00 to ietf-01:

- o Added URI mapping proposals to Section 4 as per the Email proposals to WG mailing list from Esko.

Authors' Addresses

Angelo P. Castellani
University of Padova
Via Gradenigo 6/B
Padova 35131
Italy

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal H3A 3G4
Canada

Phone: +1 514 585 0761
Email: Akbar.Rahman@InterDigital.com

Thomas Fossati
Nokia
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: thomas.fossati@nokia.com

Esko Dijk
Philips Lighting
High Tech Campus 7
Eindhoven 5656 AE
The Netherlands

Email: esko.dijk@philips.com

CoRE
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

Z. Shelby
ARM
M. Vial
Schneider-Electric
M. Koster
ARM
October 19, 2015

Reusable Interface Definitions for Constrained RESTful Environments
draft-ietf-core-interfaces-04

Abstract

This document defines a set of reusable REST resource design patterns suitable for use in constrained environments, based on IETF CoRE standards for information representation and information exchange.

Interface types for Sensors, Actuators, Parameters, and resource Collections are defined using the "if" link attribute defined by CoRE Link Format [RFC6690]. Clients may use the "if" attribute to determine how to consume resources.

Dynamic linking of state updates between resources, either on an endpoint or between endpoints, is defined with the concept of Link Bindings. We also define conditional observation attributes that work with Link Bindings or with simple CoAP Observe [RFC7641].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Interface Types	5
4. Collections	5
4.1. Introduction to Collections	5
4.2. Use Cases for Collections	6
4.3. Content-Formats for Collections	7
4.4. Links and Items in Collections	7
4.5. Queries on Collections	9
4.6. Observing Collections	9
4.7. Hypermedia Controls on Collections	9
4.8. Collection Types	10
4.9. The collection+senml+json Content-Format	10
5. Link Bindings and Observe Attributes	11
5.1. Format	12
5.2. Binding methods	13
5.3. Binding table	14
5.4. Resource Observation Attributes	14
6. Interface Descriptions	15
6.1. Link List	17
6.2. Batch	17
6.3. Linked Batch	18
6.4. Hypermedia Collection	19
6.5. Sensor	20
6.6. Parameter	21
6.7. Read-only Parameter	21
6.8. Actuator	21
6.9. Binding	22
6.10. Future Interfaces	23
6.11. WADL Description	23
7. Function Sets and Profiles	29

7.1. Defining a Function Set	29
7.1.1. Path template	30
7.1.2. Resource Type	30
7.1.3. Interface Description	30
7.1.4. Data type	31
7.2. Discovery	31
7.3. Versioning	31
8. Security Considerations	31
9. IANA Considerations	32
10. Acknowledgments	32
11. Changelog	32
12. References	34
12.1. Normative References	34
12.2. Informative References	34
Appendix A. Profile example	35
Authors' Addresses	36

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces.. CoRE Link-format is a standard for doing Web Linking [RFC5988] in constrained environments. SenML is a simple data model and representation format for composite and complex structured resources. CoRE Link-Format and SenML can be used by CoAP [RFC7252] or HTTP servers.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop. Machine application clients must be able to adapt to different resource organizations without advance knowledge of the specific data structures hosted by each connected thing. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by CoRE Link Format [RFC6690]. CoRE Link Format additionally defines a link attribute for Interface Type ("if") that can be used to describe the REST interface of a resource, and may include a link to a description document.

This document defines a set of Link Format compatible Interface Types for some common design patterns that enable the server side composition and organization, and client side discovery and consumption, of machine resources using Web Linking. An Interface Type may describe a resource in terms of it's associated content formats, data types, URI templates, REST methods, parameters, and responses. Basic interface types are defined for sensors, actuators, and properties. A set of collection types is defined for organizing

resources for discovery, and for various forms of bulk interaction with resource sets using typed embedding links.

This document introduces the concept of a Link Binding, which defines a new link relation type to create a dynamic link between resources over which to exchange state updates. Specifically, a Link Binding is a link for binding the state of 2 resources together such that updates to one are sent over the link to the other. CoRE Link Format representations are used to configure, inspect, and maintain Link Bindings. This document additionally defines a set of conditional Observe Attributes for use with Link Bindings and with the standalone CoRE Observe [RFC7641] method.

Interface Types may be used in the composition of Function Sets and Profiles. Function Sets and Profiles are described and an example is given of a sensor and actuator device profile using Function Sets composed from the Interface Types described in this document.

This document describes a set of Interface Types which are referenced by the "if" link attribute and used to implement reusable design patterns and functional abstractions. A client discovering the "if" link attribute will be able to consume resources based on its knowledge of the expected interface types. In this sense the Interface Type acts in a similar way as a Content-Format, but as a selector for a high level functional abstraction. Interface types may also be provided with hypermedia controls and affordances to drive client interaction using the principles of HATEOAS. In this case, the Interface Types serve as constructor templates for resource organization and hypermedia annotation.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. This specification makes use of the following additional terminology:

Interface Type: A resource attribute which describes the interface exposed by the resource in terms of content formats, REST methods, parameters, and other related characteristics.

Collection: A resource which contains set of related resources, referenced by a list of links and optionally consisting of subresources.

Link Binding: A unidirectional logical link between a source resource and a destination resource, over which state information is synchronized.

Resource Discovery: The process allowing a web client to identify resources being hosted on a web server.

Gradual Reveal: A REST design where resources are discovered progressively using Web Linking.

Function Set: A group of well-known REST resources that provides a particular service.

Profile: A group of well-known Function Sets defined by a specification.

Device: An IP smart object running a web server that hosts a group of Function Set instances from a profile.

Service Discovery: The process making it possible for a web client to automatically detect devices and Function Sets offered by these devices on a CoRE network.

3. Interface Types

An Interface Type definition may describe a resource in terms of its associated content formats, data types, URI templates, REST methods, parameters, and responses.

4. Collections

4.1. Introduction to Collections

A Collection is a resource which represents one or more related resources. Within this document, a collection refers to a collection with characteristics defined in this document. A Collection Interface Type consists of a set of links and a set of items pointed to by the links which may be sub-resources of the collection resource. The collection types described in this document are Link List, Batch, Linked Batch, and Hypermedia Collection.

The links in a collection are represented in CoRE Link-Format Content-Formats including JSON and CBOR variants, and the items in the collection may be represented by senml, including JSON and CBOR variants. In general, a collection may support items of any available Content-Format.

A particular resource item may be a member of more than one collection at a time by being linked to, but may only be a subresource of one collection.

Some collections may have pre-configured items and links, and some collections may support dynamic creation and removal of items and links. Likewise, modification of items in some collections may be permitted, and not in others.

Collections may support link embedding, which is analogous to an image tag (link) causing the image to display inline in a browser window. Resources pointed to by embedded links in collections may be interacted with using bulk operations on the collection resource. For example, performing a GET on a collection resource may return a single representation containing all of the linked resources.

Links in collections may be selected for processing by a particular request by using Query Filtering as described in CoRE Link-Format [RFC6690].

4.2. Use Cases for Collections

Collections may be used to provide gradual reveal of resources on an endpoint. There may be a small set of links at the .well-known/core location, which may in turn point to other collections of resources that represent device information, device configuration, device management, and various functional clusters of resources on the device.

A collection may provide resource encapsulation, where link embedding may be used to provide a single resource with which a client may interact to obtain a set of related resource values. For example, a collection for manufacturer parameters may consist of manufacturer name, date of manufacture, location of manufacture, and serial number resources which can be read as a single senml data object.

A collection may be used to group a set of like resources for bulk state update or actuation. For example, the brightness control resources of a number of luminaries may be grouped by linking to them in a collection. The collection type may support receiving a single update from a client and sending that update to each resource item in the collection.

Items may be sub-resources of the collection resource. This enables updates to multiple items in the collection to be processed together within the context of the collection resource.

Items may be dynamically created in a collection along with their hyperlinks. This provides an "item factory" pattern which can serve as a resource creation mechanism for dynamic resources. This pattern is also useful for creating temporary resources for the implementation of dynamic phenomena like commands, actions, and events using REST design patterns. Item creation uses the collection Content-Format which allows specification of links and item state in a single representation.

4.3. Content-Formats for Collections

The collection interfaces by default use CoRE Link-Format for the link representations and SenML or text/plain for representations of items. The examples given are for collections that expose resources and links in these formats. In addition, a new "collection" Content-Format is defined based on the SenML framework which represents both links and items in the collection.

The choice of whether to return a representation of the links or of the items or of the collection format is determined by the accepts header option in the request. Likewise, the choice of updating link metadata or item data or the collection resource itself is determined by the Content-Format option in the header of the update request operation.

The default Content-Formats for collection types described in this document are:

Links: application/link-format, application/link-format+json

Items: application/senml+json, text/plain

Collection: application/collection+senml+json

4.4. Links and Items in Collections

Links use CoRE Link-Format representation by default and may point to any resource reachable from the context of the collection. This includes absolute links and links that point to other network locations if the context of the collection allows. Links to sub-resources in the collection MUST have a path-element starting with the resource name, as per RFC3986 [RFC3986]. Links to resources in the global context MUST start with a root path identifier [RFC5988]. Links to other collections are formed per RFC3986.

Examples of links:

`</sen/>;if="core.lb"` : Link to the `/sen/` collection describing it as a `core.lb` type collection (Linked Batch)

`</sen/>;rel="grp"` : Link to the `/sen/` collection indicating that `/sen/` is a member of a group in the collection in which the link appears.

`<"/sen/temp">;rt="temperature"` : An absolute link to the resource at the path `/sen/temp`

`<temp>;rt="temperature"` : Link to the `temp` subresource of the collection in which this link appears.

`<temp>;anchor="/sen/"` : A link to the `temp` subresource of the collection `/sen/` which is assumed not to be a subresource of the collection in which the link appears ,but is expected to be identified in the collection by resource name.

Links in the collection MAY be Read, Updated, Added, or Removed using the CoRE Link-Format or JSON Merge-Patch Content-Formats on the collection resource. Reading links uses the GET method and returns an array or list containing the link-values of all selected links. Links may be added to the collection using POST or PATCH methods. Updates to links MUST use the PATCH method and MAY use query filtering to select links for updating. The PATCH method on links MUST use the JSON Merge-Patch Content-Format (`application/merge-patch+json`) specified in RFC7396 [RFC7396] .

Items in the collection SHOULD be represented using the SenML (`application/senml+json`) or plain text (`text/plain`) Content-Formats, depending on whether the representation is of a single data point or multiple data points. Items MAY be represented using any supported Content-Format.

Link Embedding enables the bulk processing of items in the collection using a single operation targeting the collection resource. A subset of resources in the collection may be selected for operation using Query Filtering. Bulk Read operations using GET return a SenML representation of all selected resources. Bulk item Update operations using PUT or POST apply the payload document to all selected resource items in the collection, using either a Batch or Group update policy. A Batch update is performed by applying the resource values in the payload document to all resources in the collection that match any resource name in the payload document. Group updates are performed by applying the payload document to each item in the collection. Group updates are indicated by the link relation type `rel="grp"` in the link.

The collection resource SHOULD be represented using the collection+senml+json Content-Format. The Hypermedia Collection type is the only collection type which supports this representation. Reading a collection using this content-format returns a representation of the links and the items in the collection. Performing a POST operation using this Content-Format MAY create one or more new item(s) and their corresponding links in the collection. Performing a PUT operation on this resource replaces the entire set of links and items with the payload. This Content-Format is described in section Section 4.9. Implementations MAY provide an alternate method using POST in a Content-Format used by the items in the collection which creates a default link-value and system-assigned resource name. Such implementations MAY create sub-resources of the collection resource.

4.5. Queries on Collections

Collections MAY support query filtering as defined in CoRE Link-Format [RFC6690]. Operations targeting either the links or the items MAY select a subset of links and items in the collection by using query filtering. The Content-Format specified in the request header selects whether links or items are targeted by the operation.

4.6. Observing Collections

Resource Observation using CoAP [RFC7252] MAY be supported on items in a collection. A subset of the conditional observe parameters MAY be specified to apply. In most cases pmin and pmax are useful. Resource observation on a collection's items resource MAY report any changes of resource state in any item in the collection. Observation Responses, or notifications, SHOULD provide representations of the resources that have changed in SenML Content-Format. Notifications MAY include multiple observations of a particular resource, with SenML time stamps indicating the observation times.

4.7. Hypermedia Controls on Collections

Additional Hypermedia controls may be defined to enable clients to automatically consume the collection resources. Typically, the developer may map application level semantics onto collection operations. For example, invoking an Action on an actuator may be defined as creating an Action item resource in a collection of Actions associated with the actuator, each item in the collection representing a past, current, or future action to be processed by the actuator. Removing the item could cancel any pending or current long-running action, and removing a completed action could free up resources for new actions to be invoked. A Hypermedia control for this pattern might provide a semantic name for the action, for

example "Change Brightness", and might direct the client to supply a SenML representation of parameters for the action as well as provide instructions on what method (POST) to use and how to construct the URI (the collection URI in this case) if required. An example of this hypermedia control is shown below.

4.8. Collection Types

There are four collection types defined in this document:

Collection Type	if=	Content-Formats
Link List	core.ll	link-format
Batch	core.b	link-format, senml
Linked Batch	core.lb	link-format, senml
Hypermedia	core.hc	link-format, senml,
Collection		collection+senml
Binding	core.bnd	link-format

Each collection type MAY support a subset of the methods and functions described above. For the first three collection types, the methods and functions are defined in the corresponding Interface Description. The Hypermedia Collection SHOULD expose hypermedia controls to applications to indicate which methods and functions are supported.

4.9. The collection+senml+json Content-Format

The collection+senml+json Content-Format is used to represent all of the attributes and resources of a collection in a single format. This is accomplished by extending the SenML format by adding a links element "l". The links element is formatted as an array of links in the application/link-format+json Content-Format with the tag "l" which follows the structure of the "e" element. An example of this format is given below.


```

{
  "bn": "/ep/sen/"
  "e": [
    { "n": "light", "v": 123, "u": "lx" },
    { "n": "temp", "v": 27.2, "u": "degC" },
    { "n": "humidity", "v": 80, "u": "%RH" } ],
  "l": [
    { "href": "/ep/sen/", "rel": "self", "if": "core.hc", "rt": "ms" },
    { "href": "light", "rt": "core.s" },
    { "href": "temp", "rt": "core.s" },
    { "href": "humidity", "rt": "core.s" } ]
}

```

5. Link Bindings and Observe Attributes

In a M2M RESTful environment, endpoints may directly exchange the content of their resources to operate the distributed system. For example, a light switch may supply on-off control information that may be sent directly to a light resource for on-off control. Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human intervention and a so-called commissioning tool. In this document the abstract relationship between two resources is called a link Binding. The configuration phase necessitates the exchange of binding information so a format recognized by all CoRE endpoints is essential. This document defines a format based on the CoRE Link-Format to represent binding information along with the rules to define a binding method which is a specialized relationship between two resources. The purpose of a binding is to synchronize the content between a source resource and a destination resource. The destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address). Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method. The following table gives a summary of the binding methods described in more detail in Section 5.2.

Name	Identifier	Location	Method
Polling	poll	Destination	GET
Observe	obs	Destination	GET + Observe
Push	push	Source	PUT

5.1. Format

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format are a natural way to represent binding information. This involves the creation of a new relation type, purposely named "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource. The Web link attributes allow a fine-grained control of the type of synchronization exchange along with the conditions that trigger an update. This specification defines the attributes below:

Attribute	Parameter	Value
Binding method	bind	xsd:string
Minimum Period (s)	pmin	xsd:integer (>0)
Maximum Period (s)	pmax	xsd:integer (>0)
Change Step	st	xsd:decimal (>0)
Greater Than	gt	xsd:decimal
Less Than	lt	xsd:decimal

Bind Method: This is the identifier of a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

Minimum Period: When present, the minimum period indicates the minimum time to wait (in seconds) before sending a new synchronization message (even if it has changed). In the absence of this parameter, the minimum period is up to the notifier.

Maximum Period: When present, the maximum period indicates the maximum time in seconds between two consecutive state synchronization messages (regardless if it has changed). In the absence of this parameter, the maximum period is up to the notifier. The maximum period **MUST** be greater than the minimum period parameter (if present).

Change Step: When present, the change step indicates how much the value of a resource **SHOULD** change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification **SHOULD** be between pmin and pmax.

Greater Than: When present, Greater Than indicates the upper limit value the resource value **SHOULD** cross before sending a new

notification. This parameter has lower priority than the period parameters, thus even if the Greater Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

Less Than: When present, Less Than indicates the lower limit value the resource value SHOULD cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Less Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

5.2. Binding methods

A binding method defines the rules to generate the web-transfer exchanges that will effectively send content from the source resource to the destination resource. The description of a binding method must define the following aspects:

Identifier: This is value of the "bind" attribute used to identify the method.

Location: This information indicates whether the binding entry is stored on the source or on the destination endpoint.

REST Method: This is the REST method used in the Request/Response exchanges.

Conditions: A binding method definition must state how the condition attributes of the abstract binding definition are actually used in this specialized binding.

This specification supports 3 binding methods described below.

Polling: The Polling method consists of sending periodic GET requests from the destination endpoint to the source resource and copying the content to the destination resource. The binding entry for this method MUST be stored on the destination endpoint. The destination endpoint MUST ensure that the polling frequency does not exceed the limits defined by the pmin and pmax attributes of the binding entry. The copying process MAY filter out content from the GET requests using value-based conditions (e.g Change Step, Less Than, Greater Than).

Observe: The Observe method creates an observation relationship between the destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the CoAP Observation mechanism [RFC7641]

hence this method is only permitted when the resources are made available over CoAP. The binding entry for this method MUST be stored on the destination endpoint. The binding conditions are mapped as query string parameters (see Section 5.4).

Push: When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the binding condition attributes are satisfied for the source resource. The source endpoint MUST only send a notification request if the binding conditions are met. The binding entry for this method MUST be stored on the source endpoint.

5.3. Binding table

The binding table is a special resource that gives access to the bindings on a endpoint. A binding table resource MUST support the Binding interface defined in Section 6.9. A profile SHOULD allow only one resource table per endpoint.

5.4. Resource Observation Attributes

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [RFC7641] MAY be used to observe any changes in a resource, and receive asynchronous notifications as a result. In addition, a set of query string parameters are defined here to allow a client to control how often a client is interested in receiving notifications and how much a resource value should change for the new representation to be interesting. These query parameters are described in the following table. A resource using an interface description defined in this specification and marked as Observable in its link description SHOULD support these observation parameters. The Change Step parameter can only be supported on resources with an atomic numeric value.

These query parameters MUST be treated as resources that are read using GET and updated using PUT, and MUST NOT be included in the Observe request. Multiple parameters MAY be updated at the same time by including the values in the query string of a PUT. Before being updated, these parameters have no default value.

Resource	Parameter	Data Format
Minimum Period	/ {resource} ?pmin	xsd:integer (>0)
Maximum Period	/ {resource} ?pmax	xsd:integer (>0)
Change Step	/ {resource} ?st	xsd:decimal (>0)
Less Than	/ {resource} ?lt	xsd:decimal
Greater Than	/ {resource} ?gt	xsd:decimal

Minimum Period: When present, the minimum period indicates the minimum time to wait (in seconds) before sending a new synchronization message (even if it has changed). In the absence of this parameter, the minimum period is up to the notifier.

Maximum Period: When present, the maximum period indicates the maximum time in seconds between two consecutive state synchronization messages (regardless if it has changed). In the absence of this parameter, the maximum period is up to the notifier. The maximum period **MUST** be greater than the minimum period parameter (if present).

Change Step: When present, the change step indicates how much the value of a resource **SHOULD** change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification **SHOULD** be between pmin and pmax.

Greater Than: When present, Greater Than indicates the upper limit value the resource value **SHOULD** cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Greater Than limit has been crossed, the time since the last notification **SHOULD** be between pmin and pmax.

Less Than: When present, Less Than indicates the lower limit value the resource value **SHOULD** cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Less Than limit has been crossed, the time since the last notification **SHOULD** be between pmin and pmax.

6. Interface Descriptions

This section defines REST interfaces for Link List, Batch, Sensor, Parameter, Actuator and Binding table resources. Variants such as Linked Batch or Read-Only Parameter are also presented. Each type is described along with its Interface Description attribute value and

valid methods. These are defined for each interface in the table below. These interfaces can support plain text and/or SenML Media types.

The if= column defines the Interface Description (if=) attribute value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this specification. Although these interface descriptions are intended to be used with the CoRE Link Format, they are applicable for use in any REST interface definition.

The Methods column defines the methods supported by that interface, which are described in more detail below.

Interface	if=	Methods	Content-Formats
Link List	core.ll	GET	link-format
Batch	core.b	GET, PUT, POST	link-format, senml
Linked Batch	core.lb	GET, PUT, POST, DELETE	link-format, senml
Sensor	core.s	GET	link-format, text/plain
Parameter	core.p	GET, PUT	link-format, text/plain
Read-only Parameter	core.rp	GET	link-format, text/plain
Actuator	core.a	GET, PUT, POST	link-format, text/plain
Binding	core.bnd	GET, POST, DELETE	link-format

The following is an example of links in the CoRE Link Format using these interface descriptions. The resource hierarchy is based on a simple profile defined in Appendix A. These links are used in the subsequent examples below.

```

Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s/>;rt="simple.sen";if="core.b",
</s/lt>;rt="simple.sen.lt";if="core.s",
</s/tmp>;rt="simple.sen.tmp";if="core.s";obs,
</s/hum>;rt="simple.sen.hum";if="core.s",
</a/>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d/>;rt="simple.dev";if="core.ll",
</l/>;if="core.lb",

```

6.1. Link List

The Link List interface is used to retrieve (GET) a list of resources on a web server. The GET request SHOULD contain an Accept option with the application/link-format content format; however if the resource does not support any other form of GET methods the Accept option MAY be elided. The Accept option SHOULD only include the application/link-format content format. The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on a web server.

Link List is the base interface to provide gradual reveal of resources on a CoRE web server, hence the root resource of a Function Set SHOULD implement this interface or an extension of this interface.

The following example interacts with a Link List /d containing Parameter sub-resources /d/name, /d/model.

```

Req: GET /d/ (Accept:application/link-format)
Res: 2.05 Content (application/link-format)
</d/name>;rt="simple.dev.n";if="core.p",
</d/model>;rt="simple.dev.mdl";if="core.rp"

```

6.2. Batch

The Batch interface is used to manipulate a collection of sub-resources at the same time. The Batch interface type supports the same methods as its sub-resources, and can be used to read (GET), update (PUT) or apply (POST) the values of those sub-resource with a single resource representation. The sub-resources of a Batch MAY be heterogeneous, a method used on the Batch only applies to sub-

resources that support it. For example Sensor interfaces do not support PUT, and thus a PUT request to a Sensor member of that Batch would be ignored. A batch requires the use of SenML Media types in order to support multiple sub-resources.

In addition, The Batch interface is an extension of the Link List interface and in consequence MUST support the same methods.

The following example interacts with a Batch /s/ with Sensor sub-resources /s/light, /s/temp and /s/humidity.

```
Req: GET /s/
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "light", "v": 123, "u": "lx" },
  { "n": "temp", "v": 27.2, "u": "degC" },
  { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

6.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the web server, a Linked Batch is dynamically controlled by a web client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC5988] and the CoRE Link Format [RFC6690]. A request with a POST method and a content format of application/link-format simply appends new resource links to the collection. The links in the payload MUST reference a resource on the web server with an absolute path. A DELETE request removes the entire collection. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /l/ and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.


```
Req: POST /1/ (Content-Format: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed
```

```
Req: GET /1/
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "degC" },
]
```

```
Req: POST /1/ (Content-Format: application/link-format)
</s/humidity>
Res: 2.04 Changed
```

```
Req: GET /1/ (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>
```

```
Req: GET /1/
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "degC" },
  { "n": "/s/humidity", "v": 80, "u": "%RH" }],
}
```

```
Req: DELETE /1/
Res: 2.02 Deleted
```

6.4. Hypermedia Collection

The Hypermedia Collection interface MAY provide a full set of the methods and link relation types described in section Section 4 of this document.

The following example interacts with a Hypermedia Collection at /act1/actions/ by creating a new resource with Parameter sub-resources newVal, tTime. The example depicts an actuation operation with a new actuator value of 86.3% and a transition time of 10 seconds. The returned location of the created resource is then read, and a response is returned which includes the remaining time for the operation to complete "rTime". Then, the operation is cancelled by sending a DELETE operation to the location of the created resource that represents the running action.

```
Req: POST /Act1/Actions/  
Content-Format: application/collection+senml+json  
Pl: [{"n":newVal, "v":86.3}, {"n":tTime, "v":10}]  
Res: 2.01 Created  
Location: Action1234
```

```
Req: GET /Act1/Actions/Action1234  
Accepts: application/senml+json  
Res: 2.05 Content  
Pl: [{"n":newVal, "v":86.3},  
      {"n":tTime, "v":10},  
      {"n":rTime, "v":"8.87"}]
```

```
Req: DELETE /Act1/Actions/Action1234  
Res: 2.02 Deleted
```

```
Req: GET /Act1/Actions/Action1234  
Res: 4.04 Not Found
```

6.5. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.

```
Req: GET /s/humidity (Accept: text/plain)  
Res: 2.05 Content (text/plain)  
80
```

```
Req: GET /s/humidity (Accept: application/senml+json)  
Res: 2.05 Content (application/senml+json)  
{ "e": [  
    { "n": "humidity", "v": 80, "u": "%RH" } ],  
}
```

6.6. Parameter

The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or update (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and updating a parameter.

```
Req: GET /d/name
Res: 2.05 Content (text/plain)
node5

Req: PUT /d/name (text/plain)
outdoor
Res: 2.04 Changed
```

6.7. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not updated. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

```
Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
```

6.8. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or the actuator value can be updated (PUT). In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to updated.

The following example shows requests for reading, setting and toggling an actuator (turning on a led).

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0

Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed

Req: POST /a/1/led (text/plain)
Res: 2.04 Changed

Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

6.9. Binding

The Binding interface is used to manipulate a binding table. A request with a POST method and a content format of application/link-format simply appends new bindings to the table. All links in the payload MUST have a relation type "boundTo". A GET request simply returns the current state of a binding table whereas a DELETE request empties the table.

The following example shows requests for adding, retrieving and deleting bindings in a binding table.

```
Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
Res: 2.04 Changed

Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"

Req: DELETE /bnd/
Res: 2.04 Changed
```

6.10. Future Interfaces

It is expected that further interface descriptions will be defined in this and other specifications.

6.11. WADL Description

This section defines the formal Web Application Description Language (WADL) definition of these CoRE interface descriptions.

```
<?xml version="1.0" standalone="yes"?>

<application xmlns="http://research.sun.com/wadl/2006/10"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:senml="urn:ietf:params:xml:ns:senml">

  <grammars>
    <include href="http://tools.ietf.org/html/draft-jennings-senml"/>
  </grammars>

  <doc title="CoRE Interfaces"/>

  <resource_type id="s">
    <doc title="Sensor interface type"/>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#observe-cancel"/>
    <method href="#getattr"/>
    <method href="#setattr"/>
  </resource_type>

  <resource_type id="p">
    <doc title="Parameter interface type"/>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#observe-cancel"/>
    <method href="#getattr"/>
    <method href="#setattr"/>
    <method href="#update"/>
  </resource_type>

  <resource_type id="rp">
    <doc title="Read-only Parameter interface type"/>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#observe-cancel"/>
    <method href="#getattr"/>
  </resource_type>
```

```

    <method href="#setattr"/>
</resource_type>

<resource_type id="a">
  <doc title="Actuator interface type"/>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#observe-cancel"/>
  <method href="#getattr"/>
  <method href="#setattr"/>
  <method href="#update"/>
  <method href="#apply"/>
</resource_type>

<resource_type id="ll">
  <doc title="Link List interface type"/></doc>
  <method href="#listLinks"/>
</resource_type>

<resource_type id="b">
  <doc title="Batch of sub-resources interface type">The methods read,
    observe, update and apply are applied to each sub-
    resource of the requested resource that supports it. Mixed
    sub-resource types can be supported.</doc>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#observe-cancel"/>
  <method href="#getattr"/>
  <method href="#setattr"/>
  <method href="#update"/>
  <method href="#apply"/>
  <method href="#listLinks"/>
</resource_type>

<resource_type id="lb">
  <doc title="Linked Batch interface type">. The methods read,
    observableRead, update and apply are applied to each linked
    resource of the requested resource that supports it. Mixed
    linked resource types can be supported.</doc>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#observe-cancel"/>
  <method href="#getattr"/>
  <method href="#setattr"/>
  <method href="#update"/>
  <method href="#apply"/>
  <method href="#listLinks"/>
  <method href="#appendLinks"/>

```

```

    <method href="#clearLinks"/>
  </resource_type>

  <resource_type id="hc">
    <doc title="Hypermedia Collection interface type">.</doc>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#observe-cancel"/>
    <method href="#getattr"/>
    <method href="#setattr"/>
    <method href="#update"/>
    <method href="#apply"/>
    <method href="#listLinks"/>
    <method href="#appendLinks"/>
    <method href="#clearLinks"/>
    <method href="#updateLinks"/>
    <method href="#readCollection"/>
    <method href="#addItem"/>
  </resource_type>

  <resource_type id="bnd">
    <doc title="Binding table resource type">A modifiable list of
    links. Each link MUST have the relation type "boundTo".</doc>
    <method href="#listLinks"/>
    <method href="#appendLinks"/>
    <method href="#clearLinks"/>
  </resource_type>

  <method id="read" name="GET">
    <doc>Retrieve the value of a sensor, an actuator or a parameter.
    Both HTTP and CoAP support this method.</doc>
    <request>
    </request>
    <response status="200">
      <representation mediaType="text/plain"/>
      <representation mediaType="application/senml+exi"/>
      <representation mediaType="application/senml+xml"/>
      <representation mediaType="application/senml+json"/>
    </response>
    <response status="2.05">
      <representation mediaType="text/plain"/>

    <representation mediaType="application/senml+exi"/>
      <representation mediaType="application/senml+xml"/>
      <representation mediaType="application/senml+json"/>
    </response>
  </method>

```

```

<method id="observe" name="GET">
  <doc>Observe the value of a sensor, an actuator or a parameter.
    Only CoAP supports this method since it requires the CoRE
    Observe mechanism.</doc>
  <request>
    <param name="observe" style="header" type="xsd:integer">
      <option value = 0/>
    </param>
  </request>
  <response status="2.05">
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </response>
</method>

<method id="observe-cancel" name="GET">
  <doc>Cancel observation in progress.
    Only CoAP supports this method since it requires the CoRE
    Observe mechanism.</doc>
  <request>
    <param name="observe" style="header" type="xsd:integer">
      <option value = 1/>
    </param>
  </request>
  <response status="2.05">
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </response>
</method>

<method id="update" name="PUT">
  <doc>Control the actuator or update a parameter with a new value
  or command. Both HTTP and CoAP support this method.</doc>
  <request>
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="getattr" name="GET">

```



```

<doc>Retrieve the observe attributes associated with a resource.
  Both HTTP and CoAP support this method.</doc>
<request>
  <doc>This request MUST contain an Accept option with
  application/link-format when the resource supports
  other GET methods.</doc>
  <representation mediaType="application/link-format"/>
</request>
<response status="200">
  <representation mediaType="application/link-format"/>
</response>
<response status="2.05">
  <representation mediaType="application/link-format"/>
</response>
</method>

<method id="setattr" name="PUT">
  <doc>Set the values of some or all of the observe attributes
  associated with a resource.
  Both HTTP and CoAP support this method.</doc>
  <request>
    <param name="pmin" style="query" type="xsd:integer"/>
    <param name="pmax" style="query" type="xsd:integer"/>
    <param name="lt" style="query" type="xsd:decimal"/>
    <param name="gt" style="query" type="xsd:decimal"/>
    <param name="st" style="query" type="xsd:decimal"/>
  </request>
  <response status="200">
  </response>
  <response status="2.04">
  </response>
</method>

<method id="apply" name="POST">
  <doc>Apply the value, if supplied, to resources. Both HTTP and CoAP
  support this method.</doc>
  <request>
    <doc>The apply function may contain a payload to be applied.</doc>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="listLinks" name="GET">
  <doc>Retrieve the list of Web links associated to a resource.
  Both HTTP and CoAP support this method.</doc>
  <request>
    <doc>This request MUST contain an Accept option with

```

```

    application/link-format when the resource supports
    other GET methods.</doc>
  </request>
  <response status="200">
    <representation mediaType="application/link-format"/>
  </response>
  <response status="2.05">
    <representation mediaType="application/link-format"/>
  </response>
</method>

<method id="appendLinks" name="POST">
  <doc>Append new Web links to a resource which is a collection
  of links. Both HTTP and CoAP support this method.</doc>
  <request>
    <representation mediaType="application/link-format"/>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="clearLinks" name="DELETE">
  <doc>Clear all Web Links in a resource which is a collection
  of links. Both HTTP and CoAP support this method.</doc>
  <request>
  </request>
  <response status="200"/>
  <response status="2.02"/>
</method>

<method id="updateLinks" name="PATCH">
  <doc>Update all Web Links in a resource which is a collection
  of links. Both HTTP and CoAP support this method.</doc>
  <doc>This request MUST contain a Content-Format option with
  application/merge-patch+json.</doc>
  <request>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

  <method id="addItem" name="POST">
    <doc>Add zero or more items to the collection with their links. Both HTTP and
    CoAP support this method.</doc>
    <doc>This request MAY contain a Content-Format option with
    application/collection+senml+json.</doc>
    <doc>This request MAY contain a Content-Format option with
    application/senml+json.</doc>
    <request>

```

```

    </request>
    <response status="200"/>
    <response status="2.01"/>
  </method>

  <method id="readCollection" name="GET">
    <doc>Return a representation of both links and items in the collection. Both
    HTTP and CoAP support this method.</doc>
    <doc>This request MUST contain an Accepts option with
    application/collection+senml+json.</doc>
  <request>
  </request>
  <response status="200"/>
  <response status="2.05"/>
</method>

</application>

```

7. Function Sets and Profiles

This section defines how a set of REST resources can be created called a function set. A Function Set is similar to a function block in the sense that it consists of input, output and parameter resources and contains internal logic. A Function Set can have a subset of mandatory inputs, outputs and parameters to provide minimum interoperability. It can also be extended with manufacturer/user-specific resources. A device is composed of one or more Function Set instances.

An example of function sets can be found from the CoRE Resource Directory specification that defines REST interfaces for registration, group and lookup [I-D.ietf-core-resource-directory]. The OMA Lightweight M2M standard [REF] also defines a function set structure called an Objects that use integer path, instance and resource URI segments. OMA Objects can be defined and then registered with an OMA maintained registry [REF]. This section is simply meant as a guideline for the definition of other such REST interfaces, either custom or part of other specifications.

7.1. Defining a Function Set

In a Function Set, types of resources are defined. Each type includes a human readable name, a path template, a Resource Type for discovery, the Interface Definition and the data type and allowed values. A Function Set definition may also include a field indicating if a sub-resource is mandatory or optional.

7.1.1. Path template

A Function Set is a container resource under which its sub-resources are organized. The profile defines the path to each resource of a Function Set in a path template. The template can contain either relative paths or absolute paths depending on the profile needs. An absolute Function Set should be located at its recommended root path on a web server, however it can be located under an alternative path if necessary (for example multi-purpose devices, gateways etc.). A relative Function Set can be instantiated as many times as needed on a web server with an arbitrary root path. However some Function Sets (e.g. device description) only make sense as singletons.

The path template includes a possible index {#} parameter, and possible fixed path segments. The index {#} allows for multiple instances of this type of resource, and can be any string. The root path and the indexes are the only variable elements in a path template. All other path segments should be fixed.

7.1.2. Resource Type

Each root resource of a Function Set is assigned a Resource Type parameter, therefore making it possible to discover it. Each sub-resource of a Function Set is also assigned a Resource Type parameter. This Resource Type is used for resource discovery and is usually necessary to discover optional resources supported on a specific device. The Resource Type of a Function Set may also be used for service discovery and can be exported to DNS-SD [RFC6763] for example.

The Resource Type parameter defines the value that should be included in the rt= field of the CoRE Link Format when describing a link to this resource. The value SHOULD be in the form "namespace.type" for root resources and "namespace.type.subtype" for sub-resources. This naming convention facilitates resource type filtering with the /.well-known/core resource. However a profile could allow mixing in foreign namespace references within a Function Set to import external references from other object models (e.g. SenML and UCUM).

7.1.3. Interface Description

The Interface Description parameter defines the REST interface for that type of resource. Several base interfaces are defined in Section 6 of this document. For a given profile, the Interface Description may be inferred from the Resource Type. In that case the Interface Description MAY be elided from link descriptions of resource types defined in the profile, but should be included for custom extensions to the profile.

The root resource of a Function Set should provide a list of links to its sub-resources in order to offer gradual reveal of resources. The CoRE Link List interface defined in Section 6.1 offers this functionality so a root resource should support this interface or a derived interface like CoRE Batch (See Section 6.2).

7.1.4. Data type

The Data Type field defines the type of value (and possible range) that is returned in response to a GET for that resource or accepted with a PUT. The interfaces defined in Section 6 make use of plain text and SenML Media types for the actual format of this data. A profile may restrict the list of supported content formats for the CoRE interfaces or define new interfaces with new content types.

7.2. Discovery

A device conforming to a profile SHOULD make its resources discoverable by providing links to the resources on the path /.well-known/core as defined in [RFC6690]. All resources hosted on a device SHOULD be discoverable either with a direct link in /.well-known/core or by following successive links starting from /.well-known/core.

The root path of a Function Set instance SHOULD be directly referenced in /.well-known/core in order to offer discovery at the first discovery stage. A device with more than 10 individual resources SHOULD only expose Function Set instances in /.well-known/core to limit the size of this resource.

In addition, a device MAY register its resources to a Resource Directory using the registration interface defined in [I-D.ietf-core-resource-directory] if such a directory is available.

7.3. Versioning

A profile should track Function Set changes to avoid incompatibility issues. Evolutions in a Function Set SHOULD be backward compatible.

8. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this document. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service.

9. IANA Considerations

The interface description types defined require registration.

The new link relations type "boundto" and "grp" require registration.

10. Acknowledgments

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this document.

11. Changelog

Changes from -03 to -04

- o Fixed tickets #385 and #386
- o Changed abstract and into to better describe content
- o Focus on Interface and not function set/profiles in intro
- o Changed references from draft-core-observe to RFC7641
- o Moved Function sets and Profiles to section after Interfaces
- o Moved Observe Attributes to the Link Binding section
- o Add a Collection section to describe the collection types
- o Add the Hypermedia Collection Interface Description

Changes from -02 to -03

- o Added lt and gt to binding format section.
- o Added pmin and pmax observe parameters to Observation Attributes
- o Changed the definition of lt and gt to limit crossing.
- o Added definitions for getattr and setattr to WADL.
- o Added getattr and setattr to observable interfaces.

- o Removed query parameters from Observe definition.
- o Added observe-cancel definition to WADL and to observable interfaces.

Changes from -01 to -02

- o Updated the date and version, fixed references.
- o Removed pmin and pmax observe parameters [Ticket #336]

Changes from -00 to WG Document -01

- o Improvements to the Function Set section.

Changes from -05 to WG Document -00

- o Updated the date and version.

Changes from -04 to -05

- o Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.

Changes from -03 to -04

- o Draft refresh

Changes from -02 to -03

- o Added Bindings
- o Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- o Defined a Function Set and its guidelines.
- o Added the Link List interface.
- o Added the Linked Batch interface.
- o Improved the WADL interface definition.
- o Added a simple profile example.

12. References

12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.

12.2. Informative References

- [I-D.ietf-core-resource-directory] Shelby, Z., Kostner, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-04 (work in progress), July 2015.
- [I-D.jennings-core-senml] Jennings, C., Shelby, Z., Arkko, J., and A. Keranen, "Media Types for Sensor Markup Language (SENML)", draft-jennings-core-senml-01 (work in progress), July 2015.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<http://www.rfc-editor.org/info/rfc6763>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <<http://www.rfc-editor.org/info/rfc7396>>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

Appendix A. Profile example

The following is a short definition of simple profile. This simplistic profile is for use in the examples of this document.

Function Set	Root Path	RT	IF
Device Description	/d	simple.dev	core.ll
Sensors	/s	simple.sen	core.b
Actuators	/a	simple.act	core.b

List of Function Sets

Type	Path	RT	IF	Data Type
Name	/d/name	simple.dev.n	core.p	xsd:string
Model	/d/model	simple.dev.mdl	core.rp	xsd:string

Device Description Function Set

Type	Path	RT	IF	Data Type
Light	/s/light	simple.sen.lt	core.s	xsd:decimal (lux)
Humidity	/s/humidity	simple.sen.hum	core.s	xsd:decimal (%RH)
Temperature	/s/temp	simple.sen.tmp	core.s	xsd:decimal (degC)

Sensors Function Set

Type	Path	RT	IF	Data Type
LED	/a/{#}/led	simple.act.led	core.a	xsd:boolean

Actuators Function Set

Authors' Addresses

Zach Shelby
 ARM
 150 Rose Orchard
 San Jose 95134
 FINLAND

Phone: +1-408-203-9434
 Email: zach.shelby@arm.com

Matthieu Vial
 Schneider-Electric
 Grenoble
 FRANCE

Phone: +33 (0)47657 6522
 Email: matthieu.vial@schneider-electric.com

Michael Koster
 ARM
 150 Rose Orchard
 San Jose 95134
 USA

Email: michael.koster@arm.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 12, 2019

Z. Shelby
ARM
M. Koster
SmartThings
C. Groves

J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University
March 11, 2019

Reusable Interface Definitions for Constrained RESTful Environments
draft-ietf-core-interfaces-14

Abstract

This document defines a set of Constrained RESTful Environments (CoRE) Link Format Interface Descriptions [RFC6690] applicable for use in constrained environments. These include the: Actuator, Parameter, Read-only parameter, Sensor, Batch, Linked Batch and Link List interfaces.

The Batch, Linked Batch and Link List interfaces make use of resource collections. This document further describes how collections relate to interfaces.

Many applications require a set of interface descriptions in order provide the required functionality. This document defines an Interface Description attribute value to describe resources conforming to a particular interface.

Editor's notes:

- o The git repository for the draft is found at <https://github.com/core-wg/interfaces>

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Collections	4
3.1. Introduction to Collections	5
3.2. Use Cases for Collections	5
3.3. Collection Types	6
3.4. Content-Formats for Collections	6
3.5. Link Embedding	7
3.6. Links and Items in Collections	7
3.7. Queries on Collections	8
3.8. Observing Collections	8
4. Interface Descriptions	9
4.1. Link List	11
4.2. Batch	11
4.3. Linked Batch	12
4.4. Sensor	13
4.5. Parameter	14
4.6. Read-only Parameter	14
4.7. Actuator	14
5. Security Considerations	15
6. IANA Considerations	15
6.1. Link List	15
6.2. Batch	16
6.3. Linked Batch	16

6.4. Sensor	16
6.5. Parameter	17
6.6. Read-only parameter	17
6.7. Actuator	17
7. Acknowledgements	17
8. Contributors	18
9. Changelog	18
10. References	22
10.1. Normative References	22
10.2. Informative References	22
Appendix A. Current Usage of Interfaces	23
A.1. Constrained RESTful Environments (CoRE) Link Format (IETF)	23
A.2. Open Connectivity Foundation (OCF)	24
Authors' Addresses	24

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format is a standard for doing Web Linking [RFC8288] in constrained environments. SenML [RFC8428] is a simple data model and representation format for composite and complex structured resources. CoRE Link-Format and SenML can be used by CoAP [RFC7252] or HTTP servers.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop. Machine application clients must be able to adapt to different resource organizations without advance knowledge of the specific data structures hosted by each connected thing. The use of Web Linking for the description and discovery of resources hosted by constrained origin servers is specified by CoRE Link Format [RFC6690]. CoRE Link Format additionally defines a link attribute for interface description ("if") that can be used to describe the REST interface of a resource, and may include a link to a description document.

This document defines a set of Link Format interface descriptions for some common design patterns that enable the server side composition and organization, and client side discovery and consumption, of machine resources using Web Linking. A client discovering the "if" link attribute will be able to consume resources based on its knowledge of the expected interface types. In this sense the Interface Type acts in a similar way as a Content-Format, but as a selector for a high level functional abstraction.

An interface description describes a resource in terms of its associated content formats, data types, URI templates, REST methods, parameters, and responses. Basic interface descriptions are defined for sensors, and actuators.

A set of collection types is defined for organizing resources for discovery, and for various forms of bulk interaction with resource sets using typed embedding links.

This document first defines the concept of collection interface descriptions. It then defines a number of generic interface descriptions that may be used in constrained environments. Several of these interface descriptions utilise collections.

Whilst this document assumes the use of CoAP [RFC7252], the REST interfaces described can also be realized using HTTP [RFC7230].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document requires readers to be familiar with all the terms and concepts that are discussed in [RFC8288] and [RFC6690]. This document makes use of the following additional terminology:

Gradual Reveal: A REST design where resources are discovered progressively using Web Linking.

Interface Description: The Interface Description describes the generic REST interface to interact with a resource or a set of resources. Its use is described via the Interface Description 'if' attribute which is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource. One can think of this as describing verbs usable on a resource.

Resource Discovery: The process allowing a client to identify resources being hosted on an origin server.

3. Collections

3.1. Introduction to Collections

A Collection is a resource which represents one or more related resources. [RFC6573] describes the "item" and "collection" Link Relation. An "item" link relation identifies a member of collection. A "collection" indicates the collection that an item is a member of. For example, a collection might be a resource representing a catalog of products, while an item is a resource related to an individual product.

Section 1.2.2/[RFC6690] also describes resource collections.

This document uses the concept of "collection" and applies it to interface descriptions. A collection interface description consists of a set of links and a set of items pointed to by the links which may be sub-resources of the collection resource. The collection interface descriptions described in this document are Link List, Batch and Linked Batch.

The links in a collection are represented in CoRE Link-Format Content-Formats including JSON and CBOR variants, and the items in the collection may be represented by SenML, including JSON and CBOR variants. In general, a collection may support items of any available Content-Format.

A particular resource item may be a member of more than one collection at a time by being linked to, but may only be a subresource of one collection.

Some collections may have pre-configured items and links, and some collections may support dynamic creation and removal of items and links. Likewise, modification of items in some collections may be permitted, and not in others.

Links in collections may be selected for processing by a particular request by using Query Filtering as described in CoRE Link-Format [RFC6690].

3.2. Use Cases for Collections

Collections may be used to provide gradual reveal of resources on an endpoint. There may be a small set of links at the .well-known/core location, which may in turn point to other collections of resources that represent device information, device configuration, device management, and various functional clusters of resources on the device.

A collection may be used to group a set of like resources for bulk state update or actuation. For example, the brightness control resources of a number of luminaries may be grouped by linking to them in a collection. The collection type may support receiving a single update from a client and sending that update to each resource item in the collection.

Items may be sub-resources of the collection resource. This enables updates to multiple items in the collection to be processed together within the context of the collection resource.

3.3. Collection Types

There are three collection types defined in this document:

Collection Type	if=
Link List	core.ll
Batch	core.b
Linked Batch	core.lb

Table 1: Collection Type Summary

The interface description defined in this document offer a deeper explanation of the methods that may be applied to the three collections.

3.4. Content-Formats for Collections

The collection interfaces can use the CoRE Link-Format for the link representations and SenML or text/plain for representations of items. The examples given are for collections that expose resources and links in these formats.

The choice of whether to return a representation of the links or of the items or of the collection format is determined by the Accept header option in the request. Likewise, the choice of updating link metadata or item data or the collection resource itself is determined by the Content-Format option in the header of the update request operation.

The default Content-Formats for collection types described in this document are:

Links: application/link-format, application/link-format+json

Items: application/senml+json, text/plain

3.5. Link Embedding

Collections may provide resource encapsulation by supporting link embedding. Link embedding may be used to provide a single resource with which a client may interact to obtain a set of related resource values. This is analogous to an image tag (link) causing the image to display inline in a browser window. Link embedding enables the bulk processing of items in the collection using a single operation targeting the collection resource. Performing a GET on a collection resource may return a single representation containing all of the embedded linked resources. For example, a collection for manufacturer parameters may consist of manufacturer name, date of manufacture, location of manufacture, and serial number resources which can be read as a single SenML data object.

A subset of resources in the collection may be selected for operation using Query Filtering. Bulk Read operations using GET return a SenML representation of all selected resources. Bulk item Update operations using PUT or POST apply the payload document to all selected resource items in the collection. A Batch update is performed by applying the resource values in the payload document to all resources in the collection that match any resource name in the payload document.

3.6. Links and Items in Collections

Links use CoRE Link-Format representation by default and may point to any resource reachable from the context of the collection. This includes links to resources with absolute paths as well as links that point to other network locations, if the context of the collection allows. Links to sub-resources in the collection MUST have a path-element starting with the resource name, as per [RFC3986]. Links to resources in the global context MUST start with a root path identifier [RFC8288]. Links to other collections are formed per [RFC3986].

Examples of links:

</sen/>;if="core.lb": Link to the /sen/ collection describing it as a core.lb type collection (Linked Batch)

</sen/temp>;rt="temperature": A link to the temp resource with an absolute path.

`<temp>;rt="temperature"`: Link to the temp subresource of the collection in which this link appears.

`<temp>;anchor="/sen/"`: A link to the temp subresource of the collection /sen/ which is assumed not to be a subresource of the collection in which the link appears, but is expected to be identified in the collection by resource name.

Links in the collection MAY be Read, Updated, Added, or Removed using the CoRE Link-Format or JSON Merge-Patch Content-Formats on the collection resource. Reading links uses the GET method and returns an array or list containing the link-values of all selected links. Links may be added to the collection using POST or PATCH methods. Updates to links MUST use the PATCH method and MAY use query filtering to select links for updating. The PATCH method on links MUST use the JSON Merge-Patch Content-Format (application/merge-patch+json) specified in [RFC7396].

Items in the collection SHOULD be represented using the SenML (application/senml+json) or plain text (text/plain) Content-Formats, depending on whether the representation is of a single data point or multiple data points. Items MAY be represented using any supported Content-Format.

3.7. Queries on Collections

Collections MAY support query filtering as defined in CoRE Link-Format [RFC6690]. Operations targeting either the links or the items MAY select a subset of links and items in the collection by using query filtering. The Content-Format specified in the request header selects whether links or items are targeted by the operation.

3.8. Observing Collections

Resource Observation via [I-D.ietf-core-dynlink] using CoAP [RFC7252] MAY be supported on items in a collection. A subset of the conditional observe parameters MAY be specified to apply. In most cases pmin and pmax are useful. Resource observation on a collection's resource returns the collection representation. Observation Responses, or notifications, SHOULD provide the collection representations in SenML Content-Format. Notifications MAY include multiple observations of the collection resource, with SenML time stamps indicating the observation times.

4. Interface Descriptions

This section defines REST interfaces for Sensor, Parameter, Read-Only Parameter and Actuator resource types, in addition to the Link List, Batch and Linked Batch collection types. Each type is described along with its Interface Description attribute value, valid methods and content formats. These are shown for each interface in the table below.

The if= column defines the Interface Description (if=) attribute value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this document. Although these interface descriptions are intended to be used with the CoRE Link Format, they are applicable for use in any REST interface definition.

The Methods column defines the methods supported by that interface, which are described in more detail below.

Interface	if=	Methods	Content-Formats
Link List	core.ll	GET	link-format
Batch	core.b	GET, PUT, POST	senml
Linked Batch	core.lb	GET, PUT, POST, DELETE	link-format, senml
Sensor	core.s	GET	senml, text/plain
Parameter	core.p	GET, PUT	senml, text/plain
Read-only Parameter	core.rp	GET	senml, text/plain
Actuator	core.a	GET, PUT, POST	senml, text/plain

Table 2: Interface Description Summary

The following is an example of links in the CoRE Link Format using these interface descriptions.

```

Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s/>;rt="simple.sen";if="core.b",
</s/light>;rt="simple.sen.lt";if="core.s",
</s/temp>;rt="simple.sen.tmp";if="core.s";obs,
</s/humidity>;rt="simple.sen.hum";if="core.s",
</a/>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d/>;rt="simple.dev";if="core.ll",
</l/>;if="core.lb"

```

Figure 1: Binding Interface Example

4.1. Link List

Link List is the base interface to provide gradual reveal of resources on a CoRE origin server. It is used to retrieve (GET) a list of resources on an origin server. The GET request SHOULD contain an Accept option with the application/link-format content format. However if the resource does not support any other form of content-format the Accept option MAY be elided.

Note: The use of an Accept option with application/link-format is recommended even though it is not strictly needed for the Link List interface because this interface is extended by the batch and linked batch interfaces where different content-formats are possible.

The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on an origin server.

The following example interacts with a Link List /d/ containing Parameter sub-resources /d/name, /d/model.

```
Req: GET /d/ (Accept:application/link-format)
Res: 2.05 Content (application/link-format)
</d/name>;rt="simple.dev.n";if="core.p",
</d/model>;rt="simple.dev.mdl";if="core.rp"
```

4.2. Batch

The Batch interface is used to manipulate a collection of sub-resources at the same time. The Batch interface description supports the same methods as its sub-resources, and can be used to read (GET), update (PUT) or apply (POST) the values of those sub-resource with a single resource representation. The sub-resources of a Batch MAY be heterogeneous. Hence, a method used on the Batch only applies to sub-resources that support it. For example Sensor interfaces do not support PUT, and thus a PUT request to a Sensor member of that Batch would be ignored. A batch requires the use of SenML Media types in order to support multiple sub-resources.

The following example interacts with a Batch /s/ with Sensor sub-resources /s/light, /s/temp and /s/humidity.

```
Req: GET /s/  
Res: 2.05 Content (application/senml+json)  
[  
  { "bn": "example.com/s/" },  
  { "n": "light", "v": 123, "u": "lx" },  
  { "n": "temp", "v": 27.2, "u": "Cel" },  
  { "n": "humidity", "v": 80, "u": "%RH" }  
]
```

4.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the origin server, a Linked Batch is dynamically controlled by a client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC8288] and the CoRE Link Format [RFC6690]. A request with a POST method and a content format of application/link-format simply appends new resource links to the collection. The links in the payload MUST reference a resource on the origin server with an absolute path. A DELETE request removes the entire collection. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /l/ and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.

```
Req: POST /1/ (Content-Format: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed
```

```
Req: GET /1/
Res: 2.05 Content (application/senml+json)
[
  { "bn": "example.com/" },
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "Cel" }
]
```

```
Req: POST /1/ (Content-Format: application/link-format)
</s/humidity>
Res: 2.04 Changed
```

```
Req: GET /1/ (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>
```

```
Req: GET /1/
Res: 2.05 Content (application/senml+json)
[
  { "bn": "example.com/" },
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "Cel" },
  { "n": "/s/humidity", "v": 80, "u": "%RH" }
]
```

```
Req: DELETE /1/
Res: 2.02 Deleted
```

4.4. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.

```
Req: GET /s/humidity (Accept: text/plain)
Res: 2.05 Content (text/plain)
80
```

```
Req: GET /s/humidity (Accept: application/senml+json)
Res: 2.05 Content (application/senml+json)
[
  { "bn": "example.com/s/" },
  { "n": "humidity", "v": 80, "u": "%RH" }
]
```

4.5. Parameter

The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or update (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and updating a parameter.

```
Req: GET /d/name
Res: 2.05 Content (text/plain)
node5
```

```
Req: PUT /d/name (text/plain)
outdoor
Res: 2.04 Changed
```

4.6. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not updated. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

```
Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
```

4.7. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or the actuator value can be updated

(PUT). In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to updated.

The following example shows requests for reading, setting and toggling an actuator (turning on a LED).

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

```
Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed
```

```
Req: POST /a/1/led (text/plain)
Res: 2.04 Changed
```

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

5. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this document. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service. Conversely, a malicious client could attempt to write to arbitrary resources on a poorly implemented server described in a linked batch.

6. IANA Considerations

This document registers the following CoRE Interface Description (if=) Link Target Attribute Values.

6.1. Link List

Attribute Value: core.ll

Description: The Link List interface is used to retrieve a list of resources on an origin server.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.2. Batch

Attribute Value: core.b

Description: The Batch interface is used to manipulate a collection of sub-resources at the same time.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.3. Linked Batch

Attribute Value: core.lb

Description: The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the origin server, a Linked Batch is dynamically controlled by a client.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.4. Sensor

Attribute Value: core.s

Description: The Sensor interface allows the value of a sensor resource to be read.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.5. Parameter

Attribute Value: core.p

Description: The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read or update.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.6. Read-only parameter

Attribute Value: core.rp

Description: The Read-only Parameter interface allows configuration parameters to be read but not updated.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.7. Actuator

Attribute Value: core.a

Description: The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read or the actuator value can be updated. In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle between its possible values.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

7. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further

requirements for interface descriptions have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this document. Ari Keraenen provided updated SenML examples. Christian Amsuss supplied a comprehensive review of draft -12.

8. Contributors

Matthieu Vial
Schneider-Electric
Grenoble
France

Phone: +33 (0)47657 6522
EMail: matthieu.vial@schneider-electric.com

9. Changelog

Changes from -13 to -14:

- o Version update, with changes in editor's contact information

Changes from -12 to -13:

- o SenML examples now use the Base Name (bn) labels from RFC 8428
- o Security considerations discusses client misuse of linked batches

Changes from -11 to -12:

- o Removed all text referring to function sets/profiles
- o Clarified list collections
- o Content-formats for collections and items rectified
- o Simplified Appendix A and removed Appendix B

Changes from -10 to -11:

- o Added a new Section 3.4 for Link Embedding
- o Updated examples in Section 3.5
- o Removed "Service Discovery" from Terminologies
- o Removed discussion of function sets

Changes from -09 to -10:

- o Section 1: Amendments to remove discussing properties. *
- o New author and editor added.

Changes from -08 to -09:

- o Section 3.6: Modified to indicate that the entire collection resource is returned.
- o General: Added editor's note with open issues.

Changes from -07 to -08:

- o Section 3.3: Modified Accepts to Accept header option.
- o Addressed the editor's note in Section 4.1 to clarify the use of the Accept option.

Changes from -06 to -07:

- o Corrected Figure 1 sub-resource names e.g. tmp to temp and hum to humidity.
- o Addressed the editor's note in Section 4.2.
- o Removed section on function sets and profiles as agreed to at the IETF#97.

Changes from -05 to -06:

- o Updated the abstract.
- o Section 1: Updated introduction.
- o Section 2: Alphabetised the order
- o Section 2: Removed the collections definition in favour of the complete definition in the collections section.
- o Removed section 3 on interfaces in favour of an updated definition in section 1.3.
- o General: Changed interface type to interface description as that is the term defined in RFC6690.
- o Removed section on future interfaces.

- o Section 8: Updated IANA considerations.
- o Added Appendix A to discuss current state of the art wrt to collections, function sets etc.

Changes from -04 to -05:

- o Removed Link Bindings and Observe attributes. This functionality is now contained in I-D.ietf-core-dynlink.
- o Hypermedia collections have been removed. This is covered in a new T2TRG draft.
- o The WADL description has been removed.
- o Fixed minor typos.
- o Updated references.

Changes from -03 to -04:

- o Fixed tickets #385 and #386.
- o Changed abstract and intro to better describe content.
- o Focus on Interface and not function set/profiles in intro.
- o Changed references from draft-core-observe to RFC7641.
- o Moved Function sets and Profiles to section after Interfaces.
- o Moved Observe Attributes to the Link Binding section.
- o Add a Collection section to describe the collection types.
- o Add the Hypermedia Collection Interface Description.

Changes from -02 to -03:

- o Added lt and gt to binding format section.
- o Added pmin and pmax observe parameters to Observation Attributes.
- o Changed the definition of lt and gt to limit crossing.
- o Added definitions for getattr and setattr to WADL.
- o Added getattr and setattr to observable interfaces.

- o Removed query parameters from Observe definition.
- o Added observe-cancel definition to WADL and to observable interfaces.

Changes from -01 to -02:

- o Updated the date and version, fixed references.
- o "Removed pmin and pmax observe parameters "[Ticket #336]"."

Changes from -00 to WG Document -01

- o Improvements to the Function Set section.

Changes from -05 to WG Document -00

- o Updated the date and version.

Changes from -04 to -05

- o Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.

Changes from -03 to -04

- o Draft refresh

Changes from -02 to -03

- o Added Bindings
- o Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- o Defined a Function Set and its guidelines.
- o Added the Link List interface.
- o Added the Linked Batch interface.
- o Improved the WADL interface definition.
- o Added a simple profile example.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

10.2. Informative References

- [I-D.ietf-core-dynlink]
Shelby, Z., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-08 (work in progress), March 2019.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-19 (work in progress), January 2019.
- [OIC-Core]
"OIC Resource Type Specification v1.1.0", 2016, <<https://openconnectivity.org/resources/specifications>>.
- [OIC-SmartHome]
"OIC Smart Home Device Specification v1.1.0", 2016, <<https://openconnectivity.org/resources/specifications>>.
- [OMA-TS-LWM2M]
"Lightweight Machine to Machine Technical Specification", 2016, <<http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0>>.
- [oneM2MTS0008]
"TS 0008 v1.3.2 CoAP Protocol Binding", 2016, <<http://www.onem2m.org/technical/published-documents>>.

- [oneM2MTS0023] "TS 0023 v2.0.0 Home Appliances Information Model and Mapping", 2016,
<<http://www.onem2m.org/technical/published-documents>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005,
<<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012,
<<https://www.rfc-editor.org/info/rfc6573>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014,
<<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014,
<<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014,
<<https://www.rfc-editor.org/info/rfc7396>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018,
<<https://www.rfc-editor.org/info/rfc8428>>.

Appendix A. Current Usage of Interfaces

Editor's note: This appendix will be removed. It is only included for information.

This appendix analyses the current landscape with regards the definition and use of collections and interfaces. This should be considered when considering the scope of this document.

A.1. Constrained RESTful Environments (CoRE) Link Format (IETF)

[RFC6690] assumes that different deployments or application domains will define the appropriate REST Interface Descriptions along with Resource Types to make discovery meaningful. It highlights that collections are often used for these interfaces.

Whilst 3.2/[RFC6690] defines a new Interface Description 'if' attribute the procedures around it are about the naming of the interface not what information should be included in the documentation about the interface.

A.2. Open Connectivity Foundation (OCF)

The OIC Core Specification [OIC-Core] most closely aligns with the work in this specification. It makes use of interface descriptions as per [RFC6690] and has registered several interface identifiers (<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#if-link-target-att-value>). These interface descriptors are similar to those defined in this specification. From a high level perspective:

```
links list:   OCF (oic.if.ll) -> IETF (core.ll)
               Note: it's called "link list" in the IETF.
linked batch: OCF (oic.if.b) -> IETF (core.lb)
read-only:    OCF (oic.if.r) -> IETF (core.rp)
read-write:   OCF (oic.if.rw) -> IETF (core.p)
actuator:     OCF (oic.if.a) -> IETF (core.a)
sensor:       OCF (oic.if.s) -> IETF (core.s)
batch:        No OCF equivalent -> IETF (core.b)
```

Some of the OCF interfaces make use of collections.

The OIC Core specification does not use the concept of function sets. It does however discuss the concept of profiles. The OCF defines two sets of documents. The core specification documents such as [OIC-Core] and vertical profile specification documents which provide specific information for specific applications. The OIC Smart Home Device Specification [OIC-SmartHome] is one such specification. It provides information on the resource model, discovery and data types.

Authors' Addresses

Zach Shelby
ARM
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smarththings.com

Christian Groves
Australia

Email: cngroves.std@gmail.com

Jintao Zhu
Huawei
No.127 Jinye Road, Huawei Base, High-Tech Development District
Xi'an, Shaanxi Province
China

Email: jintao.zhu@huawei.com

Bilhanan Silverajan (editor)
Tampere University
Kalevantie 4
Tampere FI-33100
Finland

Email: bilhanan.silverajan@tuni.fi

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2016

K. Li
Alibaba Group
A. Rahman
InterDigital
C. Bormann, Ed.
Universitaet Bremen TZI
November 01, 2015

Representing CoRE Formats in JSON and CBOR
draft-ietf-core-links-json-04

Abstract

JavaScript Object Notation, JSON (RFC7159) is a text-based data format which is popular for Web based data exchange. Concise Binary Object Representation, CBOR (RFC7049) is a binary data format which has been optimized for data exchange for the Internet of Things (IoT). For many IoT scenarios, CBOR formats will be preferred since it can help decrease transmission payload sizes as well as implementation code sizes compared to other data formats.

Web Linking (RFC5988) provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format (RFC6690). Outside of constrained environments, it may be useful to represent these collections of Web links in JSON, and similarly, inside constrained environments, in CBOR. This specification defines a common format for this.

Group Communication for the Constrained Application Protocol (RFC7390) defines a number of JSON formats for controlling communication between groups of nodes employing the Constrained Application Protocol (CoAP). In a similar vein, this specification defines CBOR variants of these formats.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Objectives	3
1.2. Terminology	4
2. Web Links in JSON and CBOR	4
2.1. Background	4
2.2. Information Model	5
2.3. Additional Encoding Step for CBOR	6
2.4. Examples	7
2.4.1. Link Format to CBOR Example	8
2.4.2. Link Format in JSON to CBOR Example	9
3. Group Communication Management Objects in CBOR	11
3.1. Background	11
3.2. Information Model	11
3.3. Mapping	11
3.4. Group Communication Example	12
4. IANA Considerations	14
5. Security Considerations	15
6. Acknowledgements	15
7. References	16
7.1. Normative References	16
7.2. Informative References	16
Appendix A. Implementation	17
Authors' Addresses	17

1. Introduction

Web Linking [RFC5988] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252].

The JavaScript Object Notation (JSON) [RFC7159] is a lightweight, text-based, language-independent data interchange format. JSON is popular in the Web development environment as it is easy for humans to read and write.

The Concise Binary Object Representation (CBOR) [RFC7049] is a binary data format which requires extremely small code size, allows very compact message representation, and provides extensibility without the need for version negotiation. CBOR is especially well suited for IoT environments because of these efficiencies.

When converting between a bespoke syntax such as that defined by [RFC6690] and JSON or CBOR, many small decisions have to be made. If left without guidance, it is likely that a number of slightly incompatible dialects will emerge. This specification defines a common approach for translating between the CoRE-specific bespoke formats, JSON and CBOR formats. Where applicable, mapping from other formats (e.g. CoRE Link Format) into JSON or CBOR is also described.

This specification defines a common format for representing CoRE Web Linking in JSON and CBOR, as well as the various JSON formats for controlling CoRE group communication [RFC7390], in CBOR.

Note that there is a separate question on how to represent Web links pointing out of JSON documents, as discussed e.g. in [MNOT11]. While there are good reasons to stay as compatible as possible to developments in this area, the present specification is solving a different problem.

1.1. Objectives

This specification has been designed based on the following objectives:

- o Canonical mapping
 - * lossless round-tripping with [RFC6690] and between JSON and CBOR
 - * but not trying for bit-preserving (DER-style) round-tripping

- o The simplest thing that could possibly work
 - * Do not cater for RFC 5988 complications caused by HTTP header character set issues [RFC2047]
- o Consider other work that has links in JSON, e.g.: JSON-LD, JSON-Reference [I-D.pbryan-zyp-json-ref]
 - * Do not introduce unmotivated differences

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

The term "byte" is used in its now customary sense as a synonym for "octet".

CoAP: Constrained Application Protocol [RFC7252]

CBOR: Concise Binary Object Representation [RFC7049]

CoRE: Constrained RESTful Environments, the field of work underlying [RFC6690], [RFC7049], [RFC7252], and [RFC7390]

IoT: Internet of Things

JSON: JavaScript Object Notation [RFC7159]

The objective of the JSON and CBOR mappings defined in this document is to contain information of the formats specified in [RFC5988] and [RFC6690]. This specification therefore uses the names of the ABNF productions used in those documents.

2. Web Links in JSON and CBOR

2.1. Background

Web Linking [RFC5988] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252] and in

conjunction with the CoRE resource directory
[I-D.ietf-core-resource-directory].

2.2. Information Model

This section discusses the information model underlying the CORE Link Format payload.

An application/link-format document is a collection of web links ("link-value"), each of which is a collection of attributes ("link-param") applied to a "URI-Reference".

We straightforwardly map:

- o the outer collection to an array of links;
- o each link to a JSON object or CBOR map, mapping attribute names to attribute values.

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a JSON name/value pair (member). The name is a string representation of the parameter or attribute name (as in "parmname"), the value is a string representation of the parameter or attribute value ("ptoken" or "quoted-string"). "quoted-string" productions are parsed (i.e, the outer quotes removed and the backslash constructions evaluated) as defined in [RFC6690] and its referenced documents, before placing them in JSON strings (where they may gain back additional decorations such as backslashes as defined in [RFC7159]).

If no attribute value ("ptoken" or "quoted-string") is present, the presence of the attribute name is indicated by using "true" as the value.

If a Link attribute ("parmname") is present more than once in a "link-value", its values are then represented as a JSON array of JSON string values; this array becomes the value of the JSON name/value pair where the attribute name is the JSON name. Attributes occurring just once MUST NOT be represented as JSON arrays but MUST be directly represented as JSON strings. (Note that [RFC6690] has cut down on the use of repeated parameter names; they are still allowed by [RFC5988] though. No attempt has been made to decode the possibly space-separated values for rt=, if=, and rel= into JSON arrays.)

The URI-Reference is represented as a name/value pair with the name "href" and the URI-Reference as the value. (Rationale: This usage is consistent with the use of "href" as a query parameter for link-

format query filtering and with link-format reserving the link parameter "href" specifically for this use [RFC6690]).

The resulting structure can be represented in CDDL [I-D.greevenbosch-appsawg-cbor-cddl] as:

```
links = [* link]
link = {
  href: tstr      ; resource URI
  * tstr => tstr / true
}
```

Figure 1: CoRE Link Format Data Model

2.3. Additional Encoding Step for CBOR

The above specification for JSON could be used as is for the CBOR encoding as well. However, to further reduce message sizes, it is beneficial to perform an extra encoding step, and encode "href" and some commonly occurring attribute names as small integers.

The substitution is summarized below:

name	encoded value
href	1
rel	2
anchor	3
rev	4
hreflang	5
media	6
title	7
type	8
rt	9
if	10
sz	11
ct	12
obs	13
ins	14
exp	15

Table 1: Integer Encoding of common attribute names

** TO DO: Is this the right list of attribute names? **

This list of substitutions is fixed by the present specification; no future expansion of the list is foreseen. "href" as well as all attribute names in this list MUST be represented by their integer substitutions and MUST NOT use the attribute name in text form.

This leads to the following CDDL representation for the CBOR encoding:

```
links = [* link]
link = {
  href: tstr      ; resource URI
  * label => tstr / true
}
label = tstr / &(
  href: 1,      rel: 2,      anchor: 3,
  rev: 4,      hreflang: 5,  media: 6,
  title: 7,    type: 8,      rt: 9,
  if: 10,     sz: 11,       ct: 12,
  obs: 13,
)
```

Figure 2: CoRE Link Format Data Model (CBOR)

2.4. Examples

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 3: Example from page 15 of [RFC6690]

The link-format document in Figure 3 becomes (321 bytes):

```
"[{"href":"/sensors","ct":"40","title":"Sensor
Index"},{"href":"/sensors/temp","rt":"temperature-
c","if":"sensor"},{"href":"/sensors/light","rt":"light-
lux","if":"sensor"},{"href":"http://www.example.com/sensors/
t123","anchor":"/sensors/
temp","rel":"describedby"},{"href":"/t","anchor":"/sensors/
temp","rel":"alternate"}] "
```

(More examples to be added.)

2.4.1. Link Format to CBOR Example

This examples shows conversion from link format to CBOR format.

The link-format document in Figure 3 becomes (in CBOR diagnostic format):

```
[{1: "/sensors", 12: "40", 7: "Sensor Index"},
 {1: "/sensors/temp", 9: "temperature-c", 10: "sensor"},
 {1: "/sensors/light", 9: "light-lux", 10: "sensor"},
 {1: "http://www.example.com/sensors/t123", 3: "/sensors/temp",
  2: "describedby"},
 {1: "/t", 3: "/sensors/temp", 2: "alternate"}]
```

or, in hexadecimal (203 bytes):

```
85          # array(number of data items:5)
a3          # map(# data item pairs:3)
01          # unsigned integer(value:1,"href")
68          # text string(8 bytes)
2f73656e736f7273  # "/sensors"
0c          # unsigned integer(value:12,"ct")
62          # text(2)
3430        # "40"
07          # unsigned integer(value:7,"title")
6c          # text string(12 bytes)
53656e736f7220496e646578  # "Sensor Index"
a3          # map(# data item pairs:3)
01          # unsigned integer(value:1,"href")
6d          # text string(13 bytes)
2f73656e736f72732f74  # "/sensors/temp"
656d70        # unsigned integer(value:9,"rt")
09          # text string(13 bytes)
6d          # "temperature-c"
74656d70657261747572  # unsigned integer(value:10,"if")
652d63        # text string(6 bytes)
0a          # "sensor"
66          # map(# data item pairs:3)
73656e736f72  # unsigned integer(value:1,"href")
a3          # text string(14 bytes)
01          # "/sensors/light"
6e          # unsigned integer(value:9,"rt")
2f73656e736f72732f6c  # text string(9 bytes)
69676874      # "light-lux"
09          # unsigned integer(value:10,"if")
6c696768742d6c7578  # "light-lux"
0a          # unsigned integer(value:10,"if")
```

```

66          # text string(6 bytes)
          73656e736f72          # "sensor"
a3          # map(# data item pairs:3)
01          # unsigned integer(value:1,"href")
78 23      # text string(35 bytes)
          687474703a2f2f777777
          2e6578616d706c652e63
          6f6d2f73656e736f7273
          2f74313233          # "http://www.example.com/sensors/t123"
03          # unsigned integer(value:3,"anchor")
6d          # text string(13 bytes)
          2f73656e736f72732f74
          656d70          # "/sensors/temp"
02          # unsigned integer(value:2,"rel")
6b          # text string(11 bytes)
          6465736372696265646279 # "describedby"
a3          # map(# data item pairs:3)
01          # unsigned integer(value:1,"href")
62          # text string(12 bytes)
          2f74          # "/t"
03          # unsigned integer(value:3,"anchor")
6d          # text string(13 bytes)
          2f73656e736f72732f74
          656d70          # "/sensors/temp"
02          # unsigned integer(value:2,"rel")
69          # text string(9 bytes)
          616c7465726e617465    # "alternate"

```

Figure 4: Web Links Encoded in CBOR

2.4.2. Link Format in JSON to CBOR Example

This examples shows conversion from link format JSON to CBOR format.

The JSON example from Section 2.4 becomes:

```

85          # array(number of data items:5)
a3          # map(# data item pairs:3)
01          # unsigned integer(value:1, "href")
68          # text string(8 bytes)
          2f73656e736f7273    # "/sensors"
0c          # unsigned integer(value:12,"ct")
18 28      # unsigned integer(value:40)
07          # unsigned integer(value:7,"title")
6c          # text string(12 bytes)
          53656e736f7220496e646578 # "Sensor Index"
a3          # map(# data item pairs:3)
01          # unsigned integer(value:1,"href")

```

```

        6d                # text string(13 bytes)
        2f73656e736f72732f74
        656d70            # "/sensors/temp"
    09                # unsigned integer(value:9,"rt")
    6d                # text string(13 bytes)
        74656d70657261747572
        652d63            # "temperature-c"
    0a                # unsigned integer(value:10,"if")
    66                # text string(6 bytes)
        73656e736f72      # "sensor"
a3                # map(# data item pairs:3)
    01                # unsigned integer(value:1,"href")
    6e                # text string(14 bytes)
        2f73656e736f72732f6c
        69676874          # "/sensors/light"
    09                # unsigned integer(value:9,"rt")
    69                # text string(9 bytes)
        6c696768742d6c7578
        0a                # unsigned integer(value:10,"if")
    66                # text string(6 bytes)
        73656e736f72      # "sensor"
a3                # map(# data item pairs:3)
    01                # unsigned integer(value:1,"href")
    78 23            # text string(35 bytes)
        687474703a2f2f777777
        2e6578616d706c652e63
        6f6d2f73656e736f7273
        2f74313233        # "http://www.example.com/sensors/t123"
    03                # unsigned integer(value:3,"anchor")
    6d                # text string(13 bytes)
        2f73656e736f72732f74
        656d70            # "/sensors/temp"
    02                # unsigned integer(value:2,"rel")
    6b                # text string(11 bytes)
        6465736372696265646279
        a3                # "describedby"
        a3                # map(# data item pairs:3)
    01                # unsigned integer(value:1,"href")
    62                # text string(12 bytes)
        2f74              # "/"
    03                # unsigned integer(value:3,"anchor")
    6d                # text string(13 bytes)
        2f73656e736f72732f74
        656d70            # "/sensors/temp"
    02                # unsigned integer(value:2,"rel")
    69                # text string(9 bytes)
        616c7465726e617465
        61                # "alternate"

```

Figure 5: Web Links Encoded in CBOR

3. Group Communication Management Objects in CBOR

3.1. Background

The CoAP Group Communications specification [RFC7390] defines group management objects in JSON format. These objects are used to represent IP multicast group information for CoAP endpoints. See [I-D.ietf-core-resource-directory] for more examples of using these objects.

3.2. Information Model

This section discusses the information model underlying the CoAP Group Communication management object payload.

A group membership JSON object contains one or more key/value pairs, and represents a single IP multicast group membership for the CoAP endpoint. Each key/value pair is encoded as a member of the JSON object, where the key is the member name and the value is the member's value.

The information model of the CoAP Group Communication management object can be summarized below:

```
collection = { * index => membership }
index = tstr .regexp "[A-Za-z0-9]{1,2}"
membership = {
  ? n: groupname,
  ? a: groupaddress,
}
groupname = tstr      ; host [ ":" port ]
groupaddress = tstr   ; IPv4address [ ":" port ]
                    ; / "[" IPv6address "]" [ ":" port ]
```

Figure 6: CoAP Group Communication Data Model

3.3. Mapping

The objective of the mapping defined in this section is to map information from the JSON formats specified in [RFC7390] into CBOR format, using the rules of Section 4.2 of [RFC7049].

3.4. Group Communication Example

```
{ "8" :{ "a": "[ff15::4200:f7fe:ed37:14ca]" },
  "11":{ "n": "sensors.floor1.west.bldg6.example.com",
        "a": "[ff15::4200:f7fe:ed37:25cb]" },
  "12":{ "n": "All-Devices.floor1.west.bldg6.example.com",
        "a": "[ff15::4200:f7fe:ed37:abcd]:4567" }
}
```

Figure 7: Example from section 2.6.2.4 of [RFC7390]

becomes:

```

a3                                     # map(3)
  61                                 # text(1)
    38                             # "8"
  a1                                 # map(1)
    61                             # text(1)
      61                           # "a"
    78 1b                         # text(27)
      5b666631353a3a343230
      303a663766653a656433
      373a313463615d               # "[ff15::4200:f7fe:ed37:14ca]"
  62                                 # text(2)
    3131                           # "11"
  a2                                 # map(2)
    61                             # text(1)
      6e                           # "n"
    78 25                         # text(37)
      73656e736f72732e666c
      6f6f72312e776573742e
      626c6467362e6578616d
      706c652e636f6d             # "sensors.floor1.west.bldg6.example.com"
    61                             # text(1)
      61                           # "a"
    78 1b                         # text(27)
      5b666631353a3a343230
      303a663766653a656433
      373a323563625d             # "[ff15::4200:f7fe:ed37:25cb]"
  62                                 # text(2)
    3132                           # "12"
  a2                                 # map(2)
    61                             # text(1)
      6e                           # "n"
    78 29                         # text(41)
      416c6c2d446576696365
      732e666c6f6f72312e77
      6573742e626c6467362e
      6578616d706c652e636f
      6d                           # "All-Devices.floor1.west.bldg6.example.com"
    61                             # text(1)
      61                           # "a"
    78 20                         # text(32)
      5b666631353a3a343230
      303a663766653a656433
      373a616263645d3a34353637 # "[ff15::4200:f7fe:ed37:abcd]:4567"

```

Figure 8: Group Communication Management Object Encoded in CBOR

TO DO: Should the IP address/port number information be represented in a more compact way?

4. IANA Considerations

This specification registers the following additional Internet Media Types:

Type name: application

Subtype name: link-format+json

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+json" media type are required to conform to the "application/json" Media Type and are therefore subject to the same encoding considerations specified in [RFC7159], Section 11.

Security considerations: As defined in this specification

Published specification: This specification.

Applications that use this media type: None currently known.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information:
Carsten Bormann <cabo@tzi.org>

Intended usage: COMMON

Change controller: IESG

and

Type name: application

Subtype name: link-format+cbor

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+cbor" media type are required to conform to the "application/cbor" Media Type and are therefore subject to the same encoding considerations specified in [RFC7049], Section 7.

Security considerations: As defined in this specification

Published specification: This specification.

Applications that use this media type: None currently known.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): CBOR

Person & email address to contact for further information:
Kepeng Li <kepeng.lkp@alibaba-inc.com>

Intended usage: COMMON

Change controller: IESG

5. Security Considerations

The security considerations of [RFC6690], [RFC7049] and [RFC7159] apply.

(TBD.)

6. Acknowledgements

(TBD.)

Special thanks to Bert Greevenbosch who was an author on the initial version of a contributing document, as well as the original author on the CDDL notation.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.

7.2. Informative References

- [I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-07 (work in progress), October 2015.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-05 (work in progress), October 2015.
- [I-D.pbryan-zyp-json-ref]
Bryan, P. and K. Zyp, "JSON Reference", draft-pbryan-zyp-json-ref-03 (work in progress), September 2012.
- [MNOT11] Nottingham, M., "Linking in JSON", November 2011, <http://www.mnot.net/blog/2011/11/25/linking_in_json>.

- [RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <<http://www.rfc-editor.org/info/rfc2047>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<http://www.rfc-editor.org/info/rfc7390>>.

Appendix A. Implementation

This appendix provides a simple reference implementation of the mapping between CoRE link format and Links-in-JSON.

(TBD - the reference implementation was used to create the above examples, but I still have to clean it up for readability and paste it in at 69 columns max.)

Authors' Addresses

Kepeng LI
Alibaba Group
Wenyixi Road, Yuhang District
Hangzhou, Zhejiang 311121
China

Email: kepeng.lkp@alibaba-inc.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal, Quebec H3A 3G4
Canada

Phone: +1-514-585-0761
Email: akbar.rahman@interdigital.com

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 30, 2018

K. Li
Alibaba Group
A. Rahman
InterDigital
C. Bormann, Ed.
Universitaet Bremen TZI
February 26, 2018

Representing Constrained RESTful Environments (CoRE) Link Format in JSON
and CBOR
draft-ietf-core-links-json-10

Abstract

JavaScript Object Notation, JSON (RFC 8259) is a text-based data format which is popular for Web based data exchange. Concise Binary Object Representation, CBOR (RFC7049) is a binary data format which has been optimized for data exchange for the Internet of Things (IoT). For many IoT scenarios, CBOR formats will be preferred since it can help decrease transmission payload sizes as well as implementation code sizes compared to other data formats.

Web Linking (RFC 8288) provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format (RFC 6690). Outside of constrained environments, it may be useful to represent these collections of Web links in JSON, and similarly, inside constrained environments, in CBOR. This specification defines a common format for this.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 30, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Objectives	3
1.2. Terminology	4
2. Web Links in JSON and CBOR	4
2.1. Background	4
2.2. Information Model	4
2.3. Additional Encoding Step for CBOR	6
2.4. Converting JSON or CBOR to Link-Format	8
2.5. Examples	9
2.5.1. Link Format to JSON Example	9
2.5.2. Link Format to CBOR Example	10
3. IANA Considerations	12
3.1. Media types	12
3.2. CoAP Content-Format Registration	13
4. Security Considerations	14
5. References	14
5.1. Normative References	14
5.2. Informative References	15
Appendix A. Reference implementation	16
Acknowledgements	19
Authors' Addresses	20

1. Introduction

Web Linking [RFC8288] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252].

The JavaScript Object Notation (JSON) [RFC8259] is a lightweight, text-based, language-independent data interchange format. JSON is popular in the Web development environment as it is easy for humans to read and write.

The Concise Binary Object Representation (CBOR) [RFC7049] is a binary data format which requires extremely small code size, allows very compact message representation, and provides extensibility without the need for version negotiation. CBOR is especially well suited for IoT environments because of these efficiencies.

When converting between a bespoke syntax such as that defined by [RFC6690] and JSON or CBOR, many small decisions have to be made. If left without guidance, it is likely that a number of slightly incompatible dialects will emerge. This specification defines a common format for representing CoRE Web Linking in JSON and CBOR.

Note that there is a separate question on how to represent Web links pointing out of JSON documents, as discussed for example in [MNOT11]. While there are good reasons to stay as compatible as possible to developments in this area, the present specification is solving a different problem.

1.1. Objectives

This specification has been designed based on the following objectives:

- o Canonical mapping
 - * lossless conversion in both directions between any pair of [RFC6690], JSON, and CBOR ("round-tripping"), unless prevented by a limitation of [RFC6690]
 - * but not attempting to ensure that a sequence of conversions from one of the formats through one or both of the others and back to the original would result in a bit-wise identical representation
- o The simplest thing that could possibly work.

While the formats defined in this document are based on the above objectives, they are general enough that they can be used for other applications of links in the Web. The same basic formats can be used for Web links that do not default to the "hosts" relation type (as is defined in [RFC6690]) and that allow percent encoding and general IRI syntax in what is an URI-Reference field in [RFC6690]. Also, specific support has been added for internationalized link attributes

such as "title*", including their language tags (while staying limited to UTF-8 as the character set).

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The term "byte" is used in its now customary sense as a synonym for "octet".

CoAP: Constrained Application Protocol [RFC7252]

CBOR: Concise Binary Object Representation [RFC7049]

CoRE: Constrained RESTful Environments, the field of work underlying [RFC6690], [RFC7049], [RFC7252], [RFC7641], [RFC7959], [RFC8075], and [RFC8323]

IoT: Internet of Things

JSON: JavaScript Object Notation [RFC8259]

The objective of the JSON and CBOR mappings defined in this document is to contain information of the formats specified in [RFC8288] and [RFC6690]. This specification therefore uses the names of the ABNF productions used in those documents.

2. Web Links in JSON and CBOR

2.1. Background

Web Linking [RFC8288] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252] and in conjunction with the CoRE resource directory [I-D.ietf-core-resource-directory].

2.2. Information Model

This section discusses the information model underlying the CORE Link Format payload.

An "application/link-format" document is a collection of Web links ("link-value"), each of which is a collection of attributes ("link-param") applied to a "URI-Reference".

We straightforwardly map:

- o the collection of Web links to a JSON or CBOR array of links;
- o each link to a JSON object or CBOR map, mapping attribute names to attribute values.

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a JSON name/value pair (member). The name is a string representation of the parameter or attribute name (as in "parmname"). The value can be a string, a language-tagged string, a boolean, or an array of these, as described below.

If the attribute value ("ptoken" or "quoted-string") is present, and a Link attribute with this name ("parmname") is present just once in the "link-value", the value is a string representation of the parameter or attribute value ("ptoken" or "quoted-string"). "quoted-string" productions are parsed (i.e, the outer quotes removed and the backslash constructions evaluated) as defined in [RFC6690] and its referenced documents, before placing them in JSON strings (in the representation of which they may gain back additional decorations such as backslashes as defined in [RFC8259]).

Attribute values represented as per [RFC8187], e.g. for the "title*" attribute, are converted in a language-tagged string; the attribute name is then represented without the "*" character. A language-tagged string is represented as a CBOR map (JSON object) that carries the language tag as the key for a single member and the attribute value in UTF-8 form as its value.

If no attribute value ("ptoken" or "quoted-string") is present, the presence of the attribute name is indicated by using the Boolean value "true" as the value.

If a Link attribute ("parmname") is present more than once in a "link-value", its values are then represented as a JSON array of JSON string values or "true"; this array becomes the value of the JSON name/value pair where the attribute name is the JSON name. Attributes occurring just once MUST NOT be represented as JSON arrays but MUST be directly represented as JSON strings or "true". (Note that [RFC6690] has cut down on the use of repeated parameter names; they are still allowed by [RFC8288] though. No attempt has been made to decode the possibly space-separated values for rt=, if=, and rel=

into JSON arrays.) Recipients MUST NOT accept documents that violate this requirement.

The URI-Reference is represented as a name/value pair with the name "href" and the URI-Reference as the value, with the latter converted to an IRI-Reference as per Section 3.2 of [RFC3987] (Rationale: The usage of "href" is consistent with the use of "href" as a query parameter for link-format query filtering and with link-format reserving the link parameter "href" specifically for this use [RFC6690]. The usage of an IRI-Reference is consistent with the mandate in [RFC6690] that percent-encoding be processed. Note that the format is able to represent IRIs the URIs for which cannot be represented in [RFC6690] as not all percent-encoded constructions are amenable to the pre-processing required by [RFC6690].)

As a convenient reference, the resulting structure can be described in CBOR Data Definition Language (CDDL) [I-D.ietf-cbor-cddl] as in Figure 1 (informative).

```
links = [* link]
link = {
  href: tstr    ; resource URI
  * tstr => value
}
value1 = tstr    ; text value -- the normal case
        / { tstr => tstr } ; language tag and value
        / true      ; no value given, just the name
value = value1
        / [2* value1 ] ; repeats for two or more
```

Figure 1: CoRE Link Format Data Model (JSON)

2.3. Additional Encoding Step for CBOR

The above specification for JSON might have been used as is for the CBOR encoding as well. However, to further reduce message sizes, an extra encoding step is performed: "href" and some commonly occurring attribute names are encoded as small integers.

The substitution is defined in Table 1:

name	encoded value	origin
href	1	[RFC6690], [RFCthis]
rel	2	[RFC5988] Section 5.3
anchor	3	[RFC5988] Section 5.2
rev	4	[RFC5988] Section 5.3
hreflang	5	[RFC5988] Section 5.4
media	6	[RFC5988] Section 5.4
title	7	[RFC5988] Section 5.4
type	8	[RFC5988] Section 5.4
rt	9	[RFC6690] Section 3.1
if	10	[RFC6690] Section 3.2
sz	11	[RFC6690] Section 3.3
ct	12	[RFC7252] Section 7.2.1
obs	13	[RFC7641] Section 6

Table 1: Integer Encoding of common attribute names

This list of substitutions is fixed by the present specification; no future expansion of the list is foreseen. "href" as well as all attribute names in this list MUST be represented by their integer substitutions and MUST NOT use the attribute name in text form. Recipients MUST NOT accept documents that violate this requirement.

As a convenient reference, the resulting structure can be described in CBOR Data Definition Language (CDDL) [I-D.ietf-cbor-cddl] as in Figure 2 (informative).

```

links = [* link]
link = {
  href => tstr      ; resource URI
  * label => value
}
href = 1
label = tstr / &(
  rel: 2,          anchor: 3,  rev: 4,
  hreflang: 5,     media: 6,   title: 7,
  type: 8,         rt: 9,      if: 10,
  sz: 11,         ct: 12,     obs: 13,
)
value1 = tstr      ; text value -- the normal case
           / { tstr => tstr } ; language tag and value
           / true    ; no value given, just the name
value = value1
           / [2* value1 ] ; repeats for two or more

```

Figure 2: CoRE Link Format Data Model (CBOR)

2.4. Converting JSON or CBOR to Link-Format

When a JSON or CBOR representation needs to be converted back to link-format, the above process is performed in inverse. Since link-format allows serializing link parameter values both in unquoted form ("ptoken") or in quoted form ("quoted-string"), a decision has to be made for each value. Where the syntax of "ptoken" does not allow the value to be represented, the quoted form clearly needs to be used. However, when both forms are possible, the decision is arbitrary. The recently republished Web Linking specification, [RFC8288], clarifies that this is indeed intended to be the case. However, previous specifications of link attributes, including those in [RFC5988] and [RFC6690], sometimes have made this decision in a specific way by only including one or the other alternative in the ABNF given for a link parameter. This requires a converter to know about all these cases, including those that have not been defined yet at the time of writing the converter. This problem becomes even harder by the fact that there is no central registry of link-attribute names.

Obviously, the conversion back to link-format needs to result in a valid link-format document. The reference implementation in Appendix A has addressed this problem with the following two rules:

- o Where a "ptoken" representation is possible, that is used instead of "quoted-string". This rule covers most of the special cases listed above.

- o As a special exception to the above rule, the four link attributes "anchor", "title", "rt", and "if" are always expressed as "quoted-string". This rule covers these specific four cases.

This set of rules is based on the hope that future definitions of link attributes will no longer hardcode one or the other serialization.

2.5. Examples

The examples in this section are based on an example on page 15 of [RFC6690] (Figure 3).

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 3: Example from page 15 of [RFC6690]

2.5.1. Link Format to JSON Example

The link-format document in Figure 3 becomes (321 bytes, line breaks shown are not part of the minimally-sized JSON document):

```
"[{\"href\":\"/sensors\",\"ct\":\"40\",\"title\":\"Sensor
Index\"},{\"href\":\"/sensors/temp\",\"rt\":\"temperature-
c\",\"if\":\"sensor\"},{\"href\":\"/sensors/light\",\"rt\":\"light-
lux\",\"if\":\"sensor\"},{\"href\":\"http://www.example.com/sensors/
t123\",\"anchor\":\"/sensors/
temp\",\"rel\":\"describedby\"},{\"href\":\"/t\",\"anchor\":\"/sensors/
temp\",\"rel\":\"alternate\"}] "
```

To demonstrate the handling of value-less and array-valued attributes, we extend the link-format example by examples of these (Figure 4; the "obs" attribute is defined in Section 6 of [RFC7641], while the "foo" attribute is for exposition only):

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor";obs,
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby";foo="bar";foo=3;ct=4711,
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 4: Example derived from page 15 of [RFC6690]

The link-format document in Figure 4 becomes the JSON document in Figure 5 (some spacing and indentation added):

```
[{"href":"/sensors","ct":"40","title":"Sensor Index"},
 {"href":"/sensors/temp","rt":"temperature-c","if":"sensor",
  "obs":true},
 {"href":"/sensors/light","rt":"light-lux","if":"sensor"},
 {"href":"http://www.example.com/sensors/t123",
  "anchor":"/sensors/temp","rel":"describedby",
  "foo":["bar","3"],"ct":"4711"},
 {"href":"/t","anchor":"/sensors/temp","rel":"alternate"}]
```

Figure 5: Example derived from page 15 of [RFC6690]

Note that the conversion is unable to convert the string-valued "ct" attribute to a number, which would be the natural type for a Content-Format value; similarly, both "foo" values are treated as strings independently of whether they are quoted or numeric in syntax.

2.5.2. Link Format to CBOR Example

This examples shows conversion from link format to CBOR format.

The link-format document in Figure 3 becomes (in CBOR diagnostic format):

```
[{1: "/sensors", 12: "40", 7: "Sensor Index"},
 {1: "/sensors/temp", 9: "temperature-c", 10: "sensor"},
 {1: "/sensors/light", 9: "light-lux", 10: "sensor"},
 {1: "http://www.example.com/sensors/t123", 3: "/sensors/temp",
  2: "describedby"},
 {1: "/t", 3: "/sensors/temp", 2: "alternate"}]
```

or, in hexadecimal (203 bytes):

```
85                                # array(number of data items:5)
  a3                              # map(# data item pairs:3)
    01                            # unsigned integer(value:1,"href")
    68                            # text string(8 bytes)
      2f73656e736f7273           # "/sensors"
    0c                            # unsigned integer(value:12,"ct")
    62                            # text(2)
      3430                       # "40"
    07                            # unsigned integer(value:7,"title")
    6c                            # text string(12 bytes)
      53656e736f7220496e646578   # "Sensor Index"
  a3                              # map(# data item pairs:3)
    01                            # unsigned integer(value:1,"href")
```

```

6d      # text string(13 bytes)
2f73656e736f72732f74
656d70  # "/sensors/temp"
09      # unsigned integer(value:9,"rt")
6d      # text string(13 bytes)
74656d70657261747572
652d63  # "temperature-c"
0a      # unsigned integer(value:10,"if")
66      # text string(6 bytes)
73656e736f72  # "sensor"
a3      # map(# data item pairs:3)
01      # unsigned integer(value:1,"href")
6e      # text string(14 bytes)
2f73656e736f72732f6c
69676874  # "/sensors/light"
09      # unsigned integer(value:9,"rt")
69      # text string(9 bytes)
6c696768742d6c7578  # "light-lux"
0a      # unsigned integer(value:10,"if")
66      # text string(6 bytes)
73656e736f72  # "sensor"
a3      # map(# data item pairs:3)
01      # unsigned integer(value:1,"href")
78 23   # text string(35 bytes)
687474703a2f2f777777
2e6578616d706c652e63
6f6d2f73656e736f7273
2f74313233  # "http://www.example.com/sensors/t123"
03      # unsigned integer(value:3,"anchor")
6d      # text string(13 bytes)
2f73656e736f72732f74
656d70  # "/sensors/temp"
02      # unsigned integer(value:2,"rel")
6b      # text string(11 bytes)
6465736372696265646279  # "describedby"
a3      # map(# data item pairs:3)
01      # unsigned integer(value:1,"href")
62      # text string(2 bytes)
2f74     # "/t"
03      # unsigned integer(value:3,"anchor")
6d      # text string(13 bytes)
2f73656e736f72732f74
656d70  # "/sensors/temp"
02      # unsigned integer(value:2,"rel")
69      # text string(9 bytes)
616c7465726e617465  # "alternate"

```

Figure 6: Web Links Encoded in CBOR

3. IANA Considerations

3.1. Media types

This specification registers the following additional Internet Media Types:

Type name: application

Subtype name: link-format+json

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+json" media type are required to conform to the "application/json" Media Type and are therefore subject to the same encoding considerations specified in [RFC8259], Section 11.

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

Applications that use this media type: Applications that interchange collections of Web links based on CoRE link format [RFC6690] in JSON.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information:
Carsten Bormann <cabo@tzi.org>

Intended usage: COMMON

Change controller: IESG

and

Type name: application

Subtype name: link-format+cbor

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+cbor" media type are required to conform to the "application/cbor" Media Type and are therefore subject to the same encoding considerations specified in [RFC7049], Section 7.

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

Applications that use this media type: Applications that interchange collections of Web links based on CoRE link format [RFC6690] in CBOR.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): CBOR

Person & email address to contact for further information:
Kepeng Li <kepeng.lkp@alibaba-inc.com>

Intended usage: COMMON

Change controller: IESG

3.2. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the above media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. The ID for "application/link-format+cbor" is assigned from the "Expert Review" (0-255) range, while the ID for "application/link-format+json" is assigned from the "IETF review" range. The assigned IDs are show in Table 2.

Media type	Coding	ID	Reference
application/link-format+cbor	-	TBD64	[RFCthis]
application/link-format+json	-	TBD504	[RFCthis]

Table 2: CoAP Content-Format IDs

4. Security Considerations

The security considerations relevant to the data model of [RFC6690], as well as those of [RFC7049] and [RFC8259] apply.

5. References

5.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <<https://www.rfc-editor.org/info/rfc3987>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8187] Reschke, J., "Indicating Character Encoding and Language for HTTP Header Field Parameters", RFC 8187, DOI 10.17487/RFC8187, September 2017, <<https://www.rfc-editor.org/info/rfc8187>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

- [RFC8288] Nottingham, M., "Web Linking", RFC 8288,
DOI 10.17487/RFC8288, October 2017,
<<https://www.rfc-editor.org/info/rfc8288>>.

5.2. Informative References

- [I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data
definition language (CDDL): a notational convention to
express CBOR data structures", draft-ietf-cbor-cddl-01
(work in progress), January 2018.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C.
Amsuess, "CoRE Resource Directory", draft-ietf-core-
resource-directory-12 (work in progress), October 2017.
- [MNOT11] Nottingham, M., "Linking in JSON", November 2011,
<http://www.mnot.net/blog/2011/11/25/linking_in_json>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988,
DOI 10.17487/RFC5988, October 2010,
<<https://www.rfc-editor.org/info/rfc5988>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
Application Protocol (CoAP)", RFC 7252,
DOI 10.17487/RFC7252, June 2014,
<<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained
Application Protocol (CoAP)", RFC 7641,
DOI 10.17487/RFC7641, September 2015,
<<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in
the Constrained Application Protocol (CoAP)", RFC 7959,
DOI 10.17487/RFC7959, August 2016,
<<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and
E. Dijk, "Guidelines for Mapping Implementations: HTTP to
the Constrained Application Protocol (CoAP)", RFC 8075,
DOI 10.17487/RFC8075, February 2017,
<<https://www.rfc-editor.org/info/rfc8075>>.

- [RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <<https://www.rfc-editor.org/info/rfc8323>>.
- [RUBY] "Information technology -- Programming languages -- Ruby", ISO/IEC 30170:2012, April 2012.

Appendix A. Reference implementation

A reference implementation of a converter from [RFC6690] link-format to JSON and CBOR (and back to link-format) in the programming language Ruby [RUBY] is reproduced below. (Note that this implementation does not handle [RFC8187]-encoded attributes.) For pretty-printing the binary CBOR, this uses the "cbor-diag" gem (Ruby library), which may need to be installed by "gem install cbor-diag".

```
# <CODE BEGINS>
require 'strscan'
require 'json'
require 'cbor-pretty'

class String
  def as_utf8
    force_encoding(Encoding::UTF_8)
  end
end

module CoRE
  module Links
    def self.map_to_true(a)
      Hash[a.map{ |t| [t, true]}]
    end

    PTOKENCHAR = %r"[\[\]\w!#-+\\-/:<-?^-\`{-~@]"
    QUOSTRCHAR = %r{(?:[^\\"\\]|\\.)} # to be used inside "
    ATTRCHAR = %r"[\w!#$&+.^`|~-]"
    MUSTBEQUOTED = map_to_true(%w{anchor title rt if})
    ANCHORNAME = "href"
    SCANATTR =
      %r{(?:#{ATTRCHAR}+)(?:=(?:#{PTOKENCHAR}+)|"#{QUOSTRCHAR}*"))?} # "

    RAWMAPPINGS = <<-DATA
href: 1,   rel: 2,   anchor: 3,
rev: 4,   hreflang: 5, media: 6,
title: 7, type: 8,   rt: 9,
if: 10,   sz: 11,    ct: 12,
```

```

obs: 13,
DATA

MAPPINGS = Hash.new {|h, k| k}

RAWMAPPINGS.scan(/([-\w+)\s*:\s*([-\w+)\s*),/) do |n, v|
  MAPPINGS[n] = Integer(v)
end

def self.parse(*args)
  WLNK.parse(*args)
end

class WLNK
  attr_accessor :resources
  def initialize(r = []) # make sure the keys are strings
    @resources = r.to_ary # make sure it's an Array
  end
  def self.parse(s, robust = true)
    wl = WLNK.new
    ss = StringScanner.new(s.as_utf8)
    ss.skip(/\s*/) if robust
    while ss.scan(%r{<([^\s]+)>})
      res = { ANCHORNAME => ss[1].as_utf8 }
      ss.skip(/\s*/) if robust
      while ss.skip(/;/)
        ss.skip(/\s*/) if robust
        unless ss.scan(SCANATTR)
          raise ArgumentError, "must have attribute behind ';'
            at: #{ss.peek(20).inspect} (byte #{ss.pos})"
        end
        key = ss[1].as_utf8
        value = ss[2] ||
          (ss[3] ? ss[3].gsub(/\\(.)/) { $1 } : true)
        if res[key]
          res[key] = Array(res[key]) << value
        else
          res[key] = value
        end
        ss.skip(/\s*/) if robust
      end
      wl.resources << res
      break unless ss.skip(/,/ )
      ss.skip(/\s*/) if robust
    end
    ss.skip(/\s*/) if robust
    raise ArgumentError, "link-format unparseable at:
      #{ss.peek(20).inspect} (byte #{ss.pos})" unless ss.eos?
  end
end

```

```

        wl
    end
    def to_json
        JSON.pretty_generate(@resources)
    end
    def to_cbor
        CBOR.encode(@resources.map { |r|
            Hash[r.map { |k, v| [MAPPINGS[k], v] }])
        })
    end
    def to_wlnk
        resources.map do |res|
            res = res.dup
            u = res.delete(ANCHORNAME)
            ["<#{u}>", *res.map { |k, v| wlnk_item(k, v) }].join(';')
        end.join(",")
    end
    private
    def wlnk_item(k, v)
        case v
        when String
            if MUSTBEQUOTED[k] || v !~ /\A#{PTOKENCHAR}+\z/
                "#{k}=\"#{v.gsub(/[\\"]/) { |x| "\\#{x}" }}\""
            else
                "#{k}=#{v}"
            end
        when Array
            v.map{ |v1| wlnk_item(k, v1) }.join(';')
        when true
            "#{k}"
        else
            fail "Don't know how to represent #{k=>v}.inspect"
        end
    end
end
end
end

lf = CoRE::Links.parse(ARGF.read)

puts lf.to_json           # JSON
puts CBOR.pretty(lf.to_cbor) # CBOR "pretty" binary form
puts lf.to_wlnk          # RFC 6690 link-format
# <CODE ENDS>

```

Acknowledgements

Special thanks to Bert Greevenbosch who was an author on the initial version of a contributing document as well as the original author on the CDDL notation.

Hannes Tschofenig made many helpful suggestions for improving this document.

Authors' Addresses

Kepeng LI
Alibaba Group
Wenyixi Road, Yuhang District
Hangzhou, Zhejiang 311121
China

Email: kepeng.lkp@alibaba-inc.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal, Quebec H3A 3G4
Canada

Phone: +1-514-585-0761
Email: akbar.rahman@interdigital.com

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
March 21, 2016

CoRE Resource Directory
draft-ietf-core-resource-directory-07

Abstract

In many M2M applications, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources. This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resources descriptions. Furthermore, new link attributes useful in conjunction with an RD are defined.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Architecture and Use Cases	5
3.1. Use Case: Cellular M2M	6
3.2. Use Case: Home and Building Automation	7
3.3. Use Case: Link Catalogues	7
4. Simple Directory Discovery	8
4.1. Finding a Directory Server	9
4.2. Third-party registration	10
5. Resource Directory Function Set	10
5.1. Discovery	10
5.2. Registration	12
5.3. Update	15
5.4. Removal	17
5.5. Read Endpoint Links	18
5.6. Update Endpoint Links	19
6. Group Function Set	22
6.1. Register a Group	22
6.2. Group Removal	24
7. RD Lookup Function Set	25
8. New Link-Format Attributes	29
8.1. Resource Instance attribute 'ins'	30
8.2. Export attribute 'exp'	30
9. DNS-SD Mapping	30
9.1. DNS-based Service discovery	31
9.2. mapping ins to <Instance>	32
9.3. Mapping rt to <ServiceType>	32
9.4. Domain mapping	33
9.5. TXT Record key=value strings	33
9.6. Importing resource links into DNS-SD	33
10. Security Considerations	34

10.1.	Endpoint Identification and Authentication	34
10.2.	Access Control	35
10.3.	Denial of Service Attacks	35
11.	IANA Considerations	35
11.1.	Resource Types	35
11.2.	Link Extension	36
11.3.	RD Parameter Registry	36
12.	Examples	37
12.1.	Lighting Installation	37
12.1.1.	Installation Characteristics	37
12.1.2.	RD entries	38
12.1.3.	DNS entries	42
12.1.4.	RD Operation	45
12.2.	OMA Lightweight M2M (LWM2M) Example	45
12.2.1.	The LWM2M Object Model	46
12.2.2.	LWM2M Register Endpoint	47
12.2.3.	Alternate Base URI	48
12.2.4.	LWM2M Update Endpoint Registration	49
12.2.5.	LWM2M De-Register Endpoint	49
13.	Acknowledgments	49
14.	Changelog	49
15.	References	52
15.1.	Normative References	52
15.2.	Informative References	53
	Authors' Addresses	54

1. Introduction

The work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g., 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. However, [RFC6690] only describes how to discover resources from the web server that hosts them by requesting `"/.well-known/core"`. In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of

resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resource descriptions. Furthermore, new link attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of these interfaces with CoAP [RFC7252], they can be applied in an equivalent manner to HTTP [RFC7230].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The term "byte" is used in its now customary sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

Resource Directory

A web entity that stores information about web resources and implements the REST interfaces defined in this specification for registration and lookup of those resources.

Domain

In the context of a Resource Directory, a domain is a logical grouping of endpoints. This specification assumes that the list of Domains supported by an RD is pre-configured by that RD. When a domain is exported to DNS, the domain value equates to the DNS domain name.

Group

In the context of a Resource Directory, a group is a logical grouping of endpoints for the purpose of group communications. All groups within a domain are unique.

Endpoint

Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the

Resource Directory. An endpoint is identified by its endpoint name, which is included during registration, and is unique within the associated domain of the registration.

3. Architecture and Use Cases

The resource directory architecture is illustrated in Figure 1. A Resource Directory (RD) is used as a repository for Web Links [RFC5988] about resources hosted on other web servers, which are called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port (called Context), thus a physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain sets of Web Links (called resource directory entries), and for clients to lookup resources from the RD or maintain groups. Endpoints themselves can also act as clients. An RD can be logically segmented by the use of Domains. The domain an endpoint is associated with can be defined by the RD or configured by an outside entity. This information hierarchy is shown in Figure 2.

Endpoints are assumed to proactively register and maintain resource directory entries on the RD, which are soft state and need to be periodically refreshed. An endpoint is provided with interfaces to register, update and remove a resource directory entry. Furthermore, a mechanism to discover an RD using the CoRE Link Format is defined. It is also possible for an RD to proactively discover Web Links from endpoints and add them as resource directory entries. A lookup interface for discovering any of the Web Links held in the RD is provided using the CoRE Link Format.

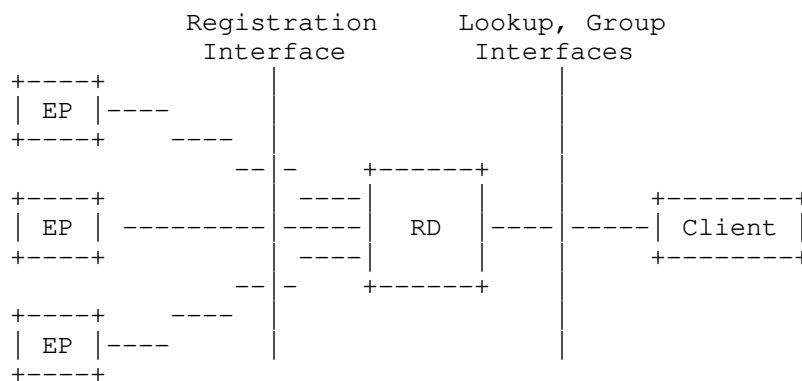


Figure 1: The resource directory architecture.

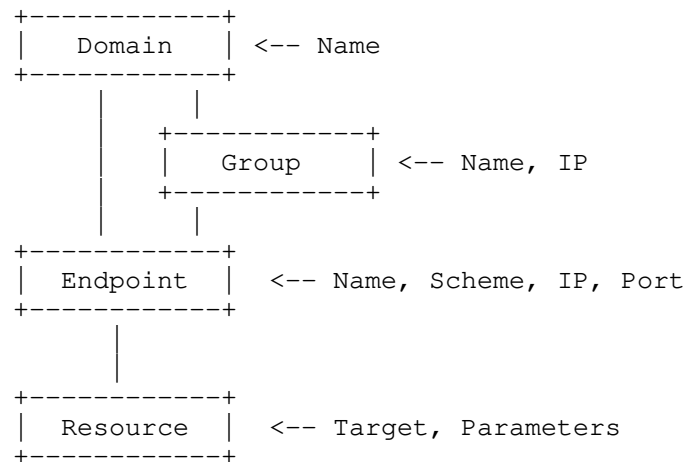


Figure 2: The resource directory information hierarchy.

3.1. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded air interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of view, the ambition is to design horizontal solutions that can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo like solutions. One of the crucial enablers of such design is the ability to discover resources (machines -- endpoints) capable of providing required information at a given time or acting on instructions from the end users.

In a typical scenario, during a boot-up procedure (and periodically afterwards), the machines (endpoints) register with a Resource Directory (for example EPs installed on vehicles enabling tracking of their position for fleet management purposes and monitoring environment parameters) hosted by the mobile operator or somewhere else in the network, periodically a description of its own capabilities. Due to the usual network configuration of mobile networks, the EPs attached to the mobile network may not always be efficiently reachable. Therefore, a remote server is usually used to provide proxy access to the EPs. The address of each (proxy) endpoint on this server is included in the resource description stored in the RD. The users, for example mobile applications for

environment monitoring, contact the RD, look-up the endpoints capable of providing information about the environment using appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server) and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look-up for EPs deployed on the vehicles the application is responsible for.

3.2. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of M2M web services. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices.

The exporting of resource information to other discovery systems is also important in these automation applications. In home automation there is a need to interact with other consumer electronics, which may already support DNS-SD, and in building automation larger resource directories or DNS-SD covering multiple buildings.

3.3. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. Resource Directory can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide the data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to a Resource Directory. Applications wishing to consume the data can use the Resource Directory lookup function set to discover and resolve links to the desired resources and endpoints. The Resource Directory service need not be coupled with the data intermediary service. Mapping of Resource Directories to data intermediaries may be many-to-many.

Metadata in web link compatible representations are supplied by Resource Directories, which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links.

External catalogs that are represented in other formats may be converted to common web linking formats for storage and access by Resource Directories. Since it is common practice for these to be URN encoded, simple and lossless structural transforms should generally be sufficient to store external metadata in Resource Directories.

The additional features of Resource Directory allow domains to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Resource groups may be defined to allow batched reads from multiple resources.

4. Simple Directory Discovery

Not all endpoints hosting resources are expected to know how to implement the Resource Directory Function Set (see Section 5) and thus explicitly register with a Resource Directory (or other such directory server). Instead, simple endpoints can implement the generic Simple Directory Discovery approach described in this section. An RD implementing this specification **MUST** implement Simple Directory Discovery. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the endpoint makes available the hosted resources that it wants to be discovered, as links on its `"/.well-known/core"` interface as specified in [RFC6690].

The endpoint then finds one or more IP addresses of the directory server it wants to know about its resources as described in Section 4.1.

An endpoint that wants to make itself discoverable occasionally sends a POST request to the `"/.well-known/core"` URI of any candidate directory server that it finds. The body of the POST request is either

- o empty, in which case the directory server is encouraged by this POST request to perform GET requests at the requesting server's default discovery URI.

or

- o a non-empty link-format document, which indicates the specific services that the requesting server wants to make known to the directory server.

The directory server integrates the information it received this way into its resource directory. It MAY make the information available to further directories, if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

The following example shows an endpoint using simple resource discovery, by simply sending a POST with its links in the body to a directory.

```

EP                                     RD
|                                     |
|  -- POST /.well-known/core "</sen/temp>..." --->  |
|                                     |
|  <---- 2.01 Created  ----->  |
|                                     |

```

4.1. Finding a Directory Server

Endpoints that want to contact a directory server can obtain candidate IP addresses for such servers in a number of ways.

In a 6LoWPAN, good candidates can be taken from:

- o specific static configuration (e.g., anycast addresses), if any,
- o the ABRO option of 6LoWPAN-ND [RFC6775],
- o other ND options that happen to point to servers (such as RDNSS),
- o DHCPv6 options that might be defined later.

In networks with more inexpensive use of multicast, the candidate IP address may be a well-known multicast address, i.e. directory servers are found by simply sending GET requests to that well-known multicast address (see Section 5.1).

As some of these sources are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

4.2. Third-party registration

For some applications, even Simple Directory Discovery may be too taxing for certain very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the Resource Directory can be filled by a third device, called an installation tool. The installation tool can fill the Resource Directory from a database or other means. For that purpose the scheme, IP address and port of the registered device is indicated in the Context parameter of the registration as well.

5. Resource Directory Function Set

This section defines the REST interfaces between an RD and endpoints, which is called the Resource Directory Function Set. Although the examples throughout this section assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. In all definitions in this section, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces defined in this section.

Resource directory entries are designed to be easily exported to other discovery mechanisms such as DNS-SD. For that reason, parameters that would meaningfully be mapped to DNS SHOULD be limited to a maximum length of 63 bytes.

5.1. Discovery

Before an endpoint can make use of an RD, it must first know the RD's IP address, port and the path of its RD Function Set. There can be several mechanisms for discovering the RD including assuming a default location (e.g. on an Edge Router in a LoWPAN), by assigning an anycast address to the RD, using DHCP, or by discovering the RD using the CoRE Link Format (see also Section 4.1). This section defines discovery of the RD using the well-known interface of the CoRE Link Format [RFC6690] as the required mechanism. It is however expected that RDs will also be discoverable via other methods depending on the deployment.

Discovery is performed by sending either a multicast or unicast GET request to `"/.well-known/core"` and including a Resource Type (rt) parameter [RFC6690] with the value `"core.rd"` in the query string. Likewise, a Resource Type parameter value of `"core.rd-lookup"` is used to discover the RD Lookup Function Set. Upon success, the response

will contain a payload with a link format entry for each RD discovered, with the URL indicating the root resource of the RD. When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network.

A Resource Directory MAY provide hints about the content-formats it supports in the links it exposes or registers, using the "ct" link attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the Resource Directory.

HTTP does not support multicast and consequently only unicast discovery can be supported using HTTP. Links to Resource Directories MAY be registered in other Resource Directories, and well-known entry points SHOULD be provided to enable the bootstrapping of unicast discovery.

An RD implementation of this specification MUST support query filtering for the rt parameter as defined in [RFC6690].

The discovery request interface is specified as follows:

Interaction: EP -> RD

Method: GET

URI Template: /.well-known/core{?rt}

URI Template Variables:

rt := Resource Type (optional). MAY contain one or more of the values "core.rd", "core.rd-lookup", "core.rd-group" or "core.rd*"

Content-Format: application/link-format (if any)

Content-Format: application/link-format+json (if any)

Content-Format: application/link-format+cbor (if any)

The following response codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format, application/link-format+json, or application/link-format+cbor payload containing one or more matching entries for the RD resource.

Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

HTTP support : NO

The following example shows an endpoint discovering an RD using this interface, thus learning that the base RD resource is, in this example, at /rd. Note that it is up to the RD to choose its base RD resource, although diagnostics and debugging is facilitated by using the base paths specified here where possible.

EP	RD
<pre> ----- GET /.well-known/core?rt=core.rd* -----> </pre>	<pre> <----- 2.05 Content "</rd>;rt="core.rd" ----- </pre>

Req: GET coap://[ff02::1]/.well-known/core?rt=core.rd*

Res: 2.05 Content

```

</rd>;rt="core.rd";ct="application/link-format+cbor",
</rd-lookup>;rt="core.rd-lookup";ct="application/link-format+cbor",
</rd-group>;rt="core.rd-group";ct="application/link-format+cbor"

```

5.2. Registration

After discovering the location of an RD Function Set, an endpoint MAY register its resources using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690], JSON CoRE Link Format (application/link-format+json), or CBOR CoRE Link Format (application/link-format+cbor) [I-D.ietf-core-links-json], along with query string parameters indicating the name of the endpoint, its domain and the lifetime of the registration. All parameters except the endpoint name are optional. It is expected that other specifications will define further parameters (see Section 11.3). The RD then creates a new resource or updates an existing resource in the RD and returns its location. An endpoint MUST use that location when refreshing

registrations using this interface. Endpoint resources in the RD are kept active for the period indicated by the lifetime parameter. The endpoint is responsible for refreshing the entry within this period using either the registration or update interface. The registration interface **MUST** be implemented to be idempotent, so that registering twice with the same endpoint parameter does not create multiple RD entries.

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: `/{"rd"}{"?ep,d,et,lt,con"}`

URI Template Variables:

`rd` := RD Function Set path (mandatory). This is the path of the RD Function Set, as obtained from discovery. An RD **SHOULD** use the value "rd" for this variable whenever possible.

`ep` := Endpoint name (mandatory). The endpoint name is an identifier that **MUST** be unique within a domain. The maximum length of this parameter is 63 bytes.

`d` := Domain (optional). The domain to which this endpoint belongs. This parameter **SHOULD** be less than 63 bytes. Optional. When this parameter is elided, the RD **MAY** associate the endpoint with a configured default domain. The domain value is needed to export the endpoint to DNS-SD (see Section 9).

`et` := Endpoint Type (optional). The semantic type of the endpoint. This parameter **SHOULD** be less than 63 bytes. Optional.

`lt` := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) **SHOULD** be assumed.

`con` := Context (optional). This parameter sets the scheme, address and port at which this server is available in the form `scheme://host:port`. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port of the register request are assumed. This parameter is mandatory when the directory is filled by a third party such as an installation tool.

Content-Format: application/link-format

Content-Format: application/link-format+json

Content-Format: application/link-format+cbor

The following response codes are defined for this interface:

Success: 2.01 "Created" or 201 "Created". The Location header MUST be included with the new resource entry for the endpoint. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration. The resource returned in the Location is only for the purpose of the Update (POST) and Removal (DELETE), and MUST NOT implement GET or PUT methods.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following example shows an endpoint with the name "node1" registering two resources to an RD using this interface. The resulting location /rd/4521 is just an example of an RD generated location.

EP		RD
---	POST /rd?ep=node1 "</sensors..." ----->	
<--	2.01 Created Location: /rd/4521 -----	

Req: POST coap://rd.example.com/rd?ep=node1

Content-Format: 40

Payload:

</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

Res: 2.01 Created

Location: /rd/4521

```
Req: POST /rd?ep=node1 HTTP/1.1
Host : example.com
Content-Format: application/link-format
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

Res: 201 Created
Location: /rd/4521
```

5.3. Update

The update interface is used by an endpoint to refresh or update its registration with an RD. To use the interface, the endpoint sends a POST request to the resource returned in the Location option in the response to the first registration. An update MAY update the lifetime or context parameters if they have changed since the last registration or update. Parameters that have not changed SHOULD NOT be included in an update. Upon receiving an update request, the RD resets the timeout for that endpoint and updates the scheme, IP address and port of the endpoint (using the source address of the update, or the context parameter if present).

An update MAY optionally add or replace links for the endpoint by including those links in the payload of the update as a CoRE Link Format document. Including links in an update message greatly increases the load on an RD and SHOULD be done infrequently. A link is replaced only if both the target URI and relation type match (see Section 10.1)

The update request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: /{+location}{?lt,con}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

con := Context (optional). This parameter sets the scheme, address and port at which this server is available in the form scheme://host:port. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port used to register are assumed. This parameter is compulsory when the directory is filled by a third party such as an installation tool.

Content-Format: application/link-format (optional)

Content-Format: application/link-format+json (optional)

Content-Format: application/link-format+cbor (optional)

The following response codes are defined for this interface:

Success: 2.04 "Changed" or 204 "No Content" if the update was successfully processed.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following example shows an endpoint updating its registration at an RD using this interface.



Req: POST /rd/4521

Res: 2.04 Changed

5.4. Removal

Although RD entries have soft state and will eventually timeout after their lifetime, an endpoint SHOULD explicitly remove its entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful earlier registration.

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

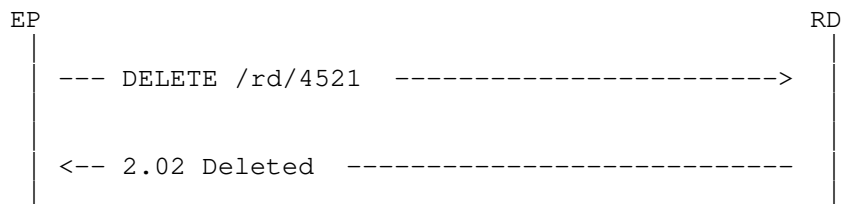
Failure: 4.00 "Bad Request" or 400 "Bad request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples shows successful removal of the endpoint from the RD.



Req: DELETE /rd/4521

Res: 2.02 Deleted

5.5. Read Endpoint Links

Some endpoints may wish to manage their links as a collection, and may need to read the current set of links in order to determine link maintenance operations.

One or more links MAY be selected by using query filtering as specified in [RFC6690] Section 4.1

The read request interface is specified as follows:

Interaction: EP -> RD

Method: GET

URI Template: /{+location}{?href,rel,rt,if,ct}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful earlier registration.

href,rel,rt,if,ct := link relations and attributes specified in the query in order to select particular links based on their relations and attributes. "href" denotes the URI target of the link. See [RFC6690] Sec. 4.1

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" upon success with an "application/link-format", "application/link-format+cbor", or "application/link-format+json" payload.

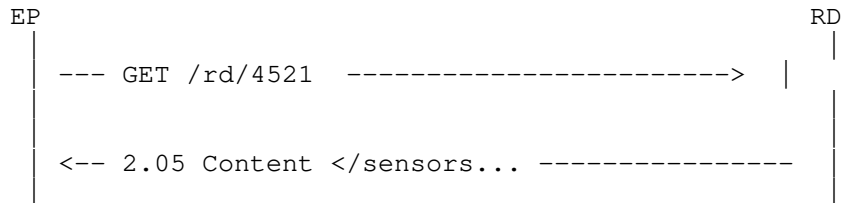
Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples show successful read of the endpoint links from the RD.



Req: GET /rd/4521

Res: 2.01 Content

Payload:

```
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"
```

5.6. Update Endpoint Links

[This section will be removed before or as a result of a working-group last-call if the PATCH methods do not achieve the same level of consensus as the present document.]

A PATCH update adds, removes or changes links for the endpoint by including link update information in the payload of the update as a merge-patch+json format [RFC7396] document.

One or more links are selected for update by using query filtering as specified in [RFC6690] Section 4.1

The query filter selects the links to be modified or deleted, by matching the query parameter values to the values of the link attributes.

When the query parameters are not present in the request, the payload specifies links to be added to the target document. When the query parameters are present, the attribute names and values in the query parameters select one or more links on which to apply the PATCH operation.

If an attribute name specified in the PATCH document exists in any the set of selected links, all occurrences of the attribute value in the target document MUST be updated using the value from the PATCH payload. If the attribute name is not present in any selected links, the attribute MUST be added to the links.

The update request interface is specified as follows:

Interaction: EP -> RD

Method: PATCH

URI Template: `/[+location]{?href,rel,rt,if,ct}`

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful earlier registration.

href,rel,rt,if,ct := link relations and attributes specified in the query in order to select particular links based on their relations and attributes. "href" denotes the URI target of the link. See [RFC6690] Sec. 4.1

Content-Format: application/merge-patch+json (mandatory)

The following response codes are defined for this interface:

Success: 2.04 "Changed" Or 204 "No Content" in the update was successfully processed.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

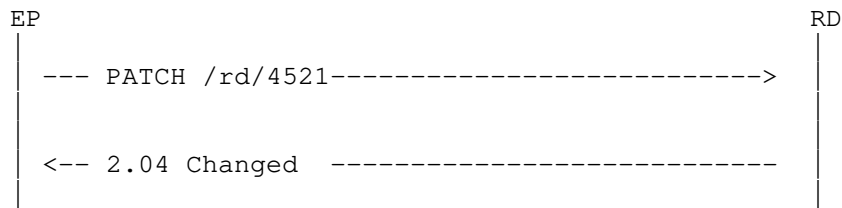
Failure: 4.04 "Not Found" or 404 "Not Found". Registration resource does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples show an endpoint adding `</sensors/humid>`, modifying `</sensors/temp>`, and removing `</sensors/light>` links in RD using the Update Endpoint Links function.

The following example shows an EP adding the link `</sensors/humid>;ct=41;rt="humidity-s";if="sensor"` to the collection of links at the location `/rd/4521`.



Req: PATCH /rd/4521

Payload:

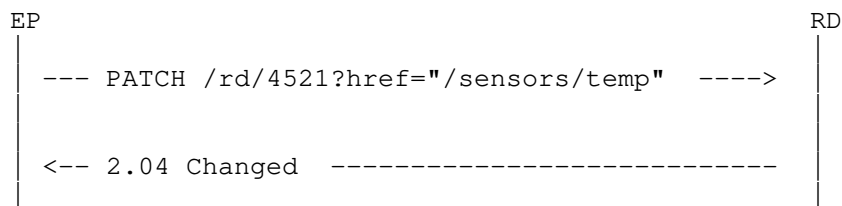
```
[{"href":"/sensors/humid","ct": 41, "rt": "humidity-s", "if": "sensor"}]
```

Content-Format:

application/merge-patch+json

Res: 2.04 Changed

The following example shows an EP modifying all links at the location /rd/4521 which are identified by href="/sensors/temp", from the initial link-value of </sensors/temp>;rt="temperature" to the new link-value </sensors/temp>;rt="temperature-c";if="sensor" by changing the value of the link attribute "rt" and adding the link attribute if="sensor" using the PATCH operation with the supplied merge-patch+json document payload.



Req: PATCH /rd/4521?href="/sensors/temp"

Payload:

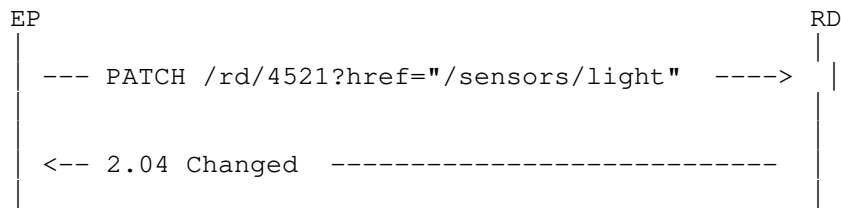
```
{"rt": "temperature-c", "if": "sensor"},
```

Content-Format:

application/merge-patch+json

Res: 2.04 Changed

This example shows an EP removing all links at the location /rd/4521 which are identified by href="/sensors/light".



Req: PATCH /rd/4521?href="/sensors/light"

Payload:
{null}

Content-Format:
application/merge-patch+json

Res: 2.04 Changed

6. Group Function Set

This section defines a function set for the creation of groups of endpoints for the purpose of managing and looking up endpoints for group operations. The group function set is similar to the resource directory function set, in that a group may be created or removed. However unlike an endpoint entry, a group entry consists of a list of endpoints and does not have a lifetime associated with it. In order to make use of multicast requests with CoAP, a group MAY have a multicast address associated with it.

6.1. Register a Group

In order to create a group, a management entity used to configure groups, makes a request to the RD indicating the name of the group to create (or update), optionally the domain the group belongs to, and optionally the multicast address of the group. The registration message includes the list of endpoints that belong to that group. If an endpoint has already registered with the RD, the RD attempts to use the context of the endpoint from its RD endpoint entry. If the client registering the group knows the endpoint has already registered, then it MAY send a blank target URI for that endpoint link when registering the group. Configuration of the endpoints themselves is out of scope of this specification. Such an interface for managing the group membership of an endpoint has been defined in [RFC7390].

The registration request interface is specified as follows:

Interaction: Manager -> RD

Method: POST

URI Template: /{+rd-group}{?gp,d,con}

URI Template Variables:

rd-group := RD Group Function Set path (mandatory). This is the path of the RD Group Function Set. An RD SHOULD use the value "rd-group" for this variable whenever possible.

gp := Group Name (mandatory). The name of the group to be created or replaced, unique within that domain. The maximum length of this parameter is 63 bytes.

d := Domain (optional). The domain to which this group belongs. The maximum length of this parameter is 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain. The domain value is needed to export the endpoint to DNS-SD (see Section 9)

con := Context (optional). This parameter is used to set the IP multicast address at which this server is available in the form scheme://multicast-address:port. Optional. In the absence of this parameter no multicast address is configured. This parameter is compulsory when the directory is filled by an installation tool.

Content-Format: application/link-format

Content-Format: application/link-format+json

Content-Format: application/link-format+cbor

The following response codes are defined for this interface:

Success: 2.01 "Created" or 201 "Created". The Location header MUST be included with the new group entry. This Location MUST be a stable identifier generated by the RD as it is used for delete operations on this registration.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following example shows an EP registering a group with the name "lights" which has two endpoints to an RD using this interface. The resulting location /rd-group/12 is just an example of an RD generated group location.

EP	RD
<pre>- POST /rd-group?gp=lights "<>;ep=node1..." --></pre>	<pre><----- 2.01 Created Location: /rd-group/12 -----</pre>

Req: POST coap://rd.example.com/rd-group?gp=lights

Payload:

```
<>;ep="node1",
<>;ep="node2"
```

Res: 2.01 Created

Location: /rd-group/12

Req: POST /rd-group?gp=lights HTTP/1.1

Host: example.com

Accept: application/link-format

Payload:

```
<>;ep="node1",
<>;ep="node2"
```

Res: 201 Created

Location: /rd-group/12

6.2. Group Removal

A group can be removed simply by sending a removal message to the location returned when registering the group. Removing a group **MUST** NOT remove the endpoints of the group from the RD.

The removal request interface is specified as follows:

Interaction: Manager -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful group registration.

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

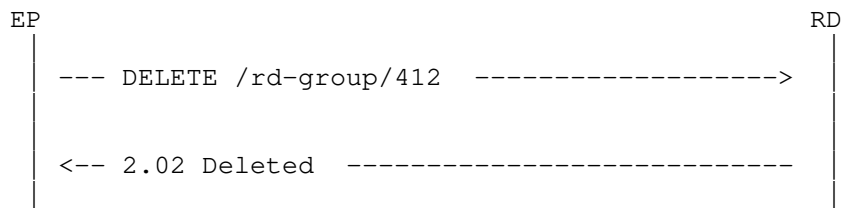
Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Group does not exist.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples shows successful removal of the group from the RD.



Req: DELETE /rd-group/12

Res: 2.02 Deleted

7. RD Lookup Function Set

In order for an RD to be used for discovering resources registered with it, a lookup interface can be provided using this function set. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. Atom or HTML Link) or using more advanced interfaces (e.g. supporting context or semantic based lookup).

This function set allows lookups for domains, groups, endpoints and resources using attributes defined in the RD Function Set and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Using the Accept Option, the requester can control whether this list is returned in CoRE Link Format ("application/link-format", default) or

its alternate content-formats ("application/link-format+json" or "application/link-format+cbor").

The target of these links SHOULD be the actual location of the domain, endpoint or resource, but MAY be an intermediate proxy e.g. in the case of an HTTP lookup interface for CoAP endpoints. Multiple query parameters MAY be included in a lookup, all included parameters MUST match for a resource to be returned. The character '*' MAY be included at the end of a parameter value as a wildcard operator.

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: /{+rd-lookup-base}/{lookup-type}{?d,ep,gp,et,rt,page,count,resource-param}

URI Template Variables:

rd-lookup-base := RD Lookup Function Set path (mandatory). This is the path of the RD Lookup Function Set. An RD SHOULD use the value "rd-lookup" for this variable whenever possible.

Content-Format: application/link-format (optional)

Content-Format: application/link-format+json (optional)

Content-Format: application/link-format+cbor (optional)

lookup-type := ("d", "ep", "res", "gp") (mandatory) This variable is used to select the kind of lookup to perform (domain, endpoint, resource, or group).

ep := Endpoint name (optional). Used for endpoint, group and resource lookups.

d := Domain (optional). Used for domain, group, endpoint and resource lookups.

page := Page (optional). Parameter can not be used without the count parameter. Results are returned from result set in pages that contains 'count' results starting from index (page * count).

count := Count (optional). Number of results is limited to this parameter value. If the parameter is not present, then an RD implementation specific default value SHOULD be used.

rt := Resource type (optional). Used for group, endpoint and resource lookups.

et := Endpoint type (optional). Used for group, endpoint and resource lookups.

resource-param := Link attribute parameters (optional). Any link attribute as defined in Section 4.1 of [RFC6690], used for resource lookups.

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format", "application/link-format+cbor", or "application/link-format+json" payload containing matching entries for the lookup.

Failure: 4.04 "Not Found" or 404 "Not Found" in case no matching entry is found for a unicast request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The examples in this section assume a CoAP host with IP address FDFD::123 and a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples. The following example shows a client performing a resource lookup:

Client	<pre> ----- GET /rd-lookup/res?rt=temperature -----> <-- 2.05 Content <coap://[FDFD::123]:61616/temp>;rt="temperature" ----- </pre>	RD
--------	--	----

Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content
 <coap://[FDFD::123]:61616/temp>;rt="temperature"

The following example shows a client performing an endpoint type lookup:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/ep?et=power-node ----->        |
|                                                         |
|<-- 2.05 Content <coap://[FDFD::123]:61616>;ep="node5" ----|
|                                                         |

```

Req: GET /rd-lookup/ep?et=power-node

Res: 2.05 Content
 <coap://[FDFD::123]:61616>;ep="node5",
 <coap://[FDFD::123]:61616>;ep="node7"

The following example shows a client performing a domain lookup:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/d ----->                        |
|                                                         |
|<-- 2.05 Content </rd>;d=domain1,</rd>;d=domain2 -----|
|                                                         |

```

Req: GET /rd-lookup/d

Res: 2.05 Content
 </rd>;d="domain1",
 </rd>;d="domain2"

The following example shows a client performing a group lookup for all groups:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/gp ----->                      |
|                                                         |
|<-- 2.05 Content </rd-group/12>;gp="lights1"; -----|
|                                     d="example.com" -----|
|                                                         |

```

Req: GET /rd-lookup/gp

Res: 2.05 Content

</rd-group/12>;gp="lights1";d="example.com"

The following example shows a client performing a lookup for all endpoints in a particular group:

Client	<pre> ----- GET /rd-lookup/ep?gp=lights1-----> <-- 2.05 Content <coap://[FDFD::123]:61616>;ep="node1" ---- </pre>	RD
--------	--	----

Req: GET /rd-lookup/ep?gp=lights1

Res: 2.05 Content

<coap://[FDFD::123]:61616>;ep="node1",
<coap://[FDFD::123]:61616>;ep="node2",

The following example shows a client performing a lookup for all groups an endpoint belongs to:

Client	<pre> ----- GET /rd-lookup/gp?ep=node1 -----> < 2.05 Content <coap://[FDFD::123]:61616>;gp="lights1"; -- ep="node1" ----- </pre>	RD
--------	--	----

Req: GET /rd-lookup/gp?ep=node1

Res: 2.05 Content

<coap://[FDFD::123]:61616>;gp="lights1";ep="node1",

8. New Link-Format Attributes

When using the CoRE Link Format to describe resources being discovered by or posted to a resource directory service, additional information about those resources is useful. This specification defines the following new attributes for use in the CoRE Link Format [RFC6690]:

```
link-extension    = ( "ins" "=" quoted-string ) ; Max 63 bytes
link-extension    = ( "exp" )
```

8.1. Resource Instance attribute 'ins'

The Resource Instance "ins" attribute is an identifier for this resource, which makes it possible to distinguish it from other similar resources. This attribute is similar in use to the <Instance> portion of a DNS-SD record (see Section 9.1, and SHOULD be unique across resources with the same Resource Type attribute in the domain it is used. A Resource Instance might be a descriptive string like "Ceiling Light, Room 3", a short ID like "AF39" or a unique UUID or iNumber. This attribute is used by a Resource Directory to distinguish between multiple instances of the same resource type within the directory.

This attribute MUST be no more than 63 bytes in length. The resource identifier attribute MUST NOT appear more than once in a link description.

8.2. Export attribute 'exp'

The Export "exp" attribute is used as a flag to indicate that a link description MAY be exported by a resource directory to external directories.

The CoRE Link Format is used for many purposes between CoAP endpoints. Some are useful mainly locally, for example checking the observability of a resource before accessing it, determining the size of a resource, or traversing dynamic resource structures. However, other links are very useful to be exported to other directories, for example the entry point resource to a functional service.

9. DNS-SD Mapping

CoRE Resource Discovery is intended to support fine-grained discovery of hosted resources, their attributes, and possibly other resource relations [RFC6690]. In contrast, service discovery generally refers to a coarse-grained resolution of an endpoint's IP address, port number, and protocol.

Resource and service discovery are complementary in the case of large networks, where the latter can facilitate scaling. This document defines a mapping between CoRE Link Format attributes and DNS-Based Service Discovery [RFC6763] fields that permits discovery of CoAP services by either method.

9.1. DNS-based Service discovery

DNS-Based Service Discovery (DNS-SD) defines a conventional method of configuring DNS PTR, SRV, and TXT resource records to facilitate discovery of services (such as CoAP servers in a subdomain) using the existing DNS infrastructure. This section gives a brief overview of DNS-SD; see [RFC6763] for a detailed specification.

DNS-SD service names are limited to 255 octets and are of the form:

Service Name = <Instance>.<ServiceType>.<Domain>.

The service name is the label of SRV/TXT resource records. The SRV RR specifies the host and the port of the endpoint. The TXT RR provides additional information in the form of key/value pairs.

The <Domain> part of the service name is identical to the global (DNS subdomain) part of the authority in URIs that identify servers or groups of servers.

The <ServiceType> part is composed of at least two labels. The first label of the pair is the application protocol name [RFC6335] preceded by an underscore character. The second label indicates the transport and is always "_udp" for UDP-based CoAP services. In cases where narrowing the scope of the search may be useful, these labels may be optionally preceded by a subtype name followed by the "_sub" label. An example of this more specific <ServiceType> is "light._sub._dali._udp".

A default <Instance> part of the service name may be set at the factory or during the commissioning process. It SHOULD uniquely identify an instance of <ServiceType> within a <Domain>. Taken together, these three elements comprise a unique name for an SRV/ TXT record pair within the DNS subdomain.

The granularity of a service name MAY be that of a host or group, or it could represent a particular resource within a CoAP server. The SRV record contains the host name (AAAA record name) and port of the service while protocol is part of the service name. In the case where a service name identifies a particular resource, the path part of the URI must be carried in a corresponding TXT record.

A DNS TXT record is in practice limited to a few hundred octets in length, which is indicated in the resource record header in the DNS response message. The data consists of one or more strings comprising a key=value pair. By convention, the first pair is txtver=<number> (to support different versions of a service description).

9.2. mapping ins to <Instance>

The Resource Instance "ins" attribute maps to the <Instance> part of a DNS-SD service name. It is stored directly in the DNS as a single DNS label of canonical precomposed UTF-8 [RFC3629] "Net-Unicode" (Unicode Normalization Form C) [RFC5198] text. However, to the extent that the "ins" attribute may be chosen to match the DNS host name of a service, it SHOULD use the syntax defined in Section 3.5 of [RFC1034] and Section 2.1 of [RFC1123].

The <Instance> part of the name of a service being offered on the network SHOULD be configurable by the user setting up the service, so that he or she may give it an informative name. However, the device or service SHOULD NOT require the user to configure a name before it can be used. A sensible choice of default name can allow the device or service to be accessed in many cases without any manual configuration at all. The default name should be short and descriptive, and MAY include a collision-resistant substring such as the lower bits of the device's MAC address, serial number, fingerprint, or other identifier in an attempt to make the name relatively unique.

DNS labels are currently limited to 63 octets in length and the entire service name may not exceed 255 octets.

9.3. Mapping rt to <ServiceType>

The resource type "rt" attribute is mapped into the <ServiceType> part of a DNS-SD service name and SHOULD conform to the reg-rel-type production of the Link Format defined in Section 2 of [RFC6690]. The "rt" attribute MUST be composed of at least a single Net-Unicode text string, without underscore '_' or period '.' and limited to 15 octets in length, which represents the application protocol name. This string is mapped to the DNS-SD <ServiceType> by prepending an underscore and appending a period followed by the "_udp" label. For example, rt="dali" is mapped into "_dali._udp".

The application protocol name may be optionally followed by a period and a service subtype name consisting of a Net-Unicode text string, without underscore or period and limited to 63 octets. This string is mapped to the DNS-SD <ServiceType> by appending a period followed by the "_sub" label and then appending a period followed by the service type label pair derived as in the previous paragraph. For example, rt="dali.light" is mapped into "light._sub._dali._udp".

The resulting string is used to form labels for DNS-SD records which are stored directly in the DNS.

9.4. Domain mapping

DNS domains may be derived from the "d" attribute. The domain attribute may be suffixed with the zone name of the authoritative DNS server to generate the domain name. The "ep" attribute is prefixed to the domain name to generate the FQDN to be stored into DNS with an AAAA RR.

9.5. TXT Record key=value strings

A number of [RFC6763] key/value pairs are derived from link-format information, to be exported in the DNS-SD as key=value strings in a TXT record ([RFC6763], Section 6.3).

The resource <URI> is exported as key/value pair "path=<URI>".

The Interface Description "if" attribute is exported as key/value pair "if=<Interface Description>".

The DNS TXT record can be further populated by importing any other resource description attributes as they share the same key=value format specified in Section 6 of [RFC6763].

9.6. Importing resource links into DNS-SD

Assuming the ability to query a Resource Directory or multicast a GET (?exp) over the local link, CoAP resource discovery may be used to populate the DNS-SD database in an automated fashion. CoAP resource descriptions (links) can be exported to DNS-SD for exposure to service discovery by using the Resource Instance attribute as the basis for a unique service name, composed with the Resource Type as the <ServiceType>, and registered in the correct <Domain>. The agent responsible for exporting records to the DNS zone file SHOULD be authenticated to the DNS server. The following example shows an agent discovering a resource to be exported:

Agent		---	GET	/rd-lookup/res?exp	----->	RD	
		<--	2.05 Content	"<coap://[FDFD::1234]:5683/light/1>;exp;			
				rt="dali.light";ins="Spot";			
				d="office";ep="node1"			

```
Req: GET /rd-lookup/res?exp

Res: 2.05 Content
<coap://[FDFD::1234]:5683/light/1>;
  exp;rt="dali.light";ins="Spot";
    d="office";ep="node1"
```

The agent subsequently registers the following DNS-SD RRs, assuming a zone name "example.com" prefixed with "office":

```
node1.office.example.com.          IN AAAA          FDFD::1234
_dali._udp.office.example.com      IN PTR
    Spot._dali._udp.office.example.com
light._sub._dali._udp.example.com  IN PTR
    Spot._dali._udp.office.example.com
Spot._dali._udp.office.example.com IN SRV    0 0 5683
    node1.office.example.com.
Spot._dali._udp.office.example.com IN TXT
    txtver=1;path=/light/1
```

In the above figure the Service Name is chosen as Spot._dali._udp.office.example.com without the light._sub service prefix. An alternative Service Name would be: Spot.light._sub._dali._udp.office.example.com.

10. Security Considerations

The security considerations as described in Section 7 of [RFC5988] and Section 6 of [RFC6690] apply. The "/.well-known/core" resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252]. DTLS or TLS based security SHOULD be used on all resource directory interfaces defined in this document.

10.1. Endpoint Identification and Authentication

An Endpoint is determined to be unique by an RD by the Endpoint identifier parameter included during Registration, and any associated TLS or DTLS security bindings. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint or Client on a resource directory SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security. Endpoints using a Certificate MUST include the Endpoint identifier as the Subject of the Certificate, and this identifier MUST be checked by a resource

directory to match the Endpoint identifier included in the Registration message.

10.2. Access Control

Access control SHOULD be performed separately for the RD Function Set and the RD Lookup Function Set, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the domain, endpoint or resource level.

10.3. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly become part of a DDoS attack as UDP does not require return routability check. Therefore, an attacker can easily spoof the source IP of the target entity and send requests to such a service which would then respond to the target entity. This can be used for large-scale DDoS attacks on the target. Especially, if the service returns a response that is order of magnitudes larger than the request, the situation becomes even worse as now the attack can be amplified. DNS servers have been widely used for DDoS amplification attacks. Recently, it has been observed that NTP Servers, that also run on unprotected UDP have been used for DDoS attacks (<http://tools.cisco.com/security/center/content/CiscoSecurityNotice/CVE-2013-5211>) since there is no return routability check and can have a large amplification factor. The responses from the NTP server were found to be 19 times larger than the request. A Resource Directory (RD) which responds to wild-card lookups is potentially vulnerable if run with CoAP over UDP. Since there is no return routability check and the responses can be significantly larger than requests, RDs can unknowingly become part of a DDoS amplification attack. Therefore, it is RECOMMENDED that implementations ensure return routability. This can be done, for example by responding to wild card lookups only over DTLS or TLS or TCP.

11. IANA Considerations

11.1. Resource Types

"core.rd", "core.rd-group" and "core.rd-lookup" resource types need to be registered with the resource type registry defined by [RFC6690].

11.2. Link Extension

The "exp" attribute needs to be registered when a future Web Linking link-extension registry is created (e.g. in RFC5988bis).

11.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include the human readable name of the parameter, the query parameter, validity requirements if any and a description. The query parameter MUST be a valid URI query key [RFC3986].

Initial entries in this sub-registry are as follows:

Name	Query	Validity	Description
Endpoint Name	ep	60-4294967295	Name of the endpoint
Lifetime	lt		Lifetime of the registration in seconds
Domain	d		Domain to which this endpoint belongs
Endpoint Type	et	URI	Semantic name of the endpoint
Context	con		The scheme, address and port at which this server is available
Endpoint Name Group Name	gp		Name of the endpoint, max 63 bytes Name of a group in the RD
Page Count	page count	Integer	Used for pagination
		Integer	Used for pagination

Table 1: RD Parameters

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC5226].

12. Examples

Examples are added here.

12.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the Resource Directory (RD) with a CoAP interface to facilitate the installation and start up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address. No conclusions must be drawn on the realization of actual installation procedures, because the example "emphasizes" some of the issues that may influence the use of the RD.

12.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one end-point. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager. Following the lay-out of cables and routers the network manager has defined DNS domains. The presence sensor and luminary1 are part of DNS domain: rtr_5612_rrt.example.com and luminary2 is part of rtr_7899_pfa.example.com. The names of

luminary1- luminary2-, and sensor- interfaces are respectively: lm_12-345-678, lm_12-456-378, and sn_12-345-781. These names are stored in DNS together with their IP addresses. The FQDN of the interfaces is shown in Table 2 below:

Name	FQDN
luminary1	lm_12-345-678.rtr_5612_rrt.example.com
luminary2	lm_12-456-378.rtr_7899_pfa.example.com
Presence sensor	sn_12-345-781.rtr_5612_rrt.example.com
Resource directory	pc_123456.rtr_5612_rrt.example.com

Table 2: interface FQDNs

At the moment of installation, the network under installation is not necessarily connected to the DNS infra structure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and sensor shown in Table 3 below:

Name	IPv6 address
luminary1	FDFD::ABCD:1
luminary2	FDFD::ABCD:2
Presence sensor	FDFD::ABCD:3
Resource directory	FDFD::ABCD:0

Table 3: interface SLAAC addresses

In Section 12.1.2 the use of resource directory during installation is presented. In Section 12.1.3 the connection to DNS is discussed.

12.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have rt: light, and the presence sensor has rt: p-sensor. The end-points have names which are relevant to the light installation manager. In this case luminary1, luminary2, and the presence sensor are located in room 2-4-015, where luminary1 is located at the window and luminary2 and the presence sensor are located at the door. The end-point names reflect this physical

location. The middle, left and right lamps are accessed via path /light/middle, /light/left, and /light/right respectively. The identifiers relevant to the Resource Directory are shown in Table 4 below:

Name	end-point	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	light
luminary1	lm_R2-4-015_wndw	/light/middle	light
luminary1	lm_R2-4-015_wndw	/light/right	light
luminary2	lm_R2-4-015_door	/light/left	light
luminary2	lm_R2-4-015_door	/light/middle	light
luminary2	lm_R2-4-015_door	/light/right	light
Presence sensor	ps_R2-4-015_door	/ps	p-sensor

Table 4: Resource Directory identifiers

The CT inserts the end-points of the luminaries and the sensor in the RD using the Context parameter (con) to specify the interface address:

```
Req: POST coap://[FDFD::ABCD:0]/rd
    ?ep=lm_R2-4-015_wndw&con=coap://[FDFD::ABCD:1]
Payload:
</light/left>;rt="light";
    d="R2-4-015";ins="lamp4444";exp,
</light/middle>;rt="light";
    d="R2-4-015";ins="lamp5555";exp,
</light/right>;rt="light";
    d="R2-4-015";ins="lamp6666";exp

Res: 2.01 Created
Location: /rd/4521
```



```
Req: POST coap://[FDFD::ABCD:0]/rd
      ?ep=lm_R2-4-015_door&con=coap://[FDFD::ABCD:2]
```

```
Payload:
```

```
</light/left>;rt="light";
  d="R2-4-015";ins="lamp1111";exp,
</light/middle>;rt="light";
  d="R2-4-015";ins="lamp2222";exp,
</light/right>;rt="light";
  d="R2-4-015";ins="lamp3333";exp
```

```
Res: 2.01 Created
Location: /rd/4522
```

```
Req: POST coap://[FDFD::ABCD:0]/rd
      ?ep=ps_R2-4-015_door&con=coap://[FDFD::ABCD:3]
```

```
Payload:
```

```
</ps>;rt="p-sensor";
  d="R2-4-015";ins="pres1234";exp
```

```
Res: 2.01 Created
Location: /rd/4523
```

The domain name d="R2-4-015" has been added for an efficient lookup because filtering on "ep" name is awkward. The same domain name is communicated to the two luminaries and the presence sensor by the CT. The "exp" attribute is set for the later administration in DNS of the instance name ins="lampxxxx".

Once the individual endpoints are registered, the group needs to be registered. Because the presence sensor sends one multicast message to the luminaries, all lamps in the group need to have an identical path. This path is created on the two luminaries using the batch command defined in [I-D.ietf-core-interfaces]. The path to a batch of lamps is defined as: /light/grp1. In the example below, two endpoints are updated with an additional resource using the path /light/grp1 on the two luminaries.

```
Req: POST
      coap://[FDFD::ABCD:1]/light/grp1
      (Content-Format:application/link-format)<light/middle>,<light/left>
```

```
Res: 2.04 Changed
```

```
Req: POST
      coap://[FDFD::ABCD:2]/light/grp1
      (Content-Format:application/link-format)<light/right>
```

```
Res: 2.04 Changed
```

The group is specified in the RD. The Context parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, these two end-points and the end-point of the presence sensor are registered as members of the group.

It is expected that Standards Developing Organizations (SDOs) may develop other special purpose protocols to specify additional group links, group membership, group names and other parameters in the individual nodes.

```
Req: POST coap://[FDFD::ABCD:0]/rd-group
?gp=grp_R2-4-015;con="coap://[FF05::1]";exp;ins="grp1234"
Payload:
<>ep=lm_R2-4-015_wndw,
<>ep=lm_R2-4-015_door,
<>ep=ps_R2-4-015_door

Res: 2.01 Created
Location: /rd-group/501
```

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its domain, queries the RD for the end-point with `rt=light` and `d=R2-4-015`. The RD returns all end-points in the domain.

```
Req: GET coap://[FDFD::ABCD:0]/rd-lookup/ep
?d=R2-4-015;rt=light

Res: 2.05 Content
<coap://[FDFD::ABCD:1]>;
  ep="lm_R2-4-015_wndw",
<coap://[FDFD::ABCD:2]>;
  ep="lm_R2-4-015_door"
```

Knowing its own IPv6 address, the luminary discovers its endpoint name. With the end-point name the luminary queries the RD for all groups to which the end-point belongs.

```
Req: GET coap://[FDFD::ABCD:0]/rd-lookup/gp
?ep=lm_R2-4-015_wndw

Res: 2.05 Content
<coap://[FF05::1]>;gp="grp_R2-4-015"
```

From the context parameter value, the luminary learns the multicast address of the multicast group.

Alternatively, the CT can communicate the multicast address directly to the luminaries by using the "coap-group" resource specified in [RFC7390].

```
Req: POST //[FDFD::ABCD:1]/coap-group
      Content-Format: application/coap-group+json
      { "a": "[FF05::1]" }
      { "n": "grp_R2-4-015" }
```

```
Res: 2.01 Created
Location-Path: /coap-group/1
```

Dependent on the situation only the address , "a", or the name, "n", is specified in the coap-group resource. Instead of the RD group name also the DNS group name can be used.

12.1.3. DNS entries

The network manager assigns the domain bc.example.com to the entries coming from the RD. The agent that looks up the resource directory uses the domain name bc.example.com as prescribed, to enter the services and hosts into the DNS.

The agent does a lookup as specified in Section 9.6. The RD returns all entries annotated with "exp". The agent subsequently registers the following DNS-SD RRs:

```

lm_R2-4-015_wndw.bc.example.com.      IN AAAA      FDFD::ABCD:1
lm_R2-4-015_door.bc.example.com.      IN AAAA      FDFD::ABCD:2
ps_R2-4-015_door.bc.example.com.      IN AAAA      FDFD::ABCD:3
_light._udp.bc.example.com            IN PTR
                                     lamp1111._light._udp.bc.example.com
_light._udp.bc.example.com            IN PTR
                                     lamp2222._light._udp.bc.example.com
_light._udp.bc.example.com            IN PTR
                                     lamp3333._light._udp.bc.example.com
_light._udp.bc.example.com            IN PTR
                                     lamp4444._light._udp.bc.example.com
_light._udp.bc.example.com            IN PTR
                                     lamp5555._light._udp.bc.example.com
_light._udp.bc.example.com            IN PTR
                                     lamp6666._light._udp.bc.example.com
_p-sensor._udp.bc.example.com          IN PTR
                                     pres12324._p-sensor._udp.bc.example.com
lamp1111._light._udp.bc.example.com    IN SRV  0 0 5683
                                     lm_R2-4-015_door.bc.example.com.
lamp1111._light._udp.bc.example.com    IN TXT
                                     txtver=1;path=/light/left
lamp2222._light._udp.bc.example.com    IN SRV  0 0 5683
                                     lm_R2-4-015_door.bc.example.com.
lamp2222._light._udp.bc.example.com    IN TXT
                                     txtver=1;path=/light/middle
lamp3333._light._udp.bc.example.com    IN SRV  0 0 5683
                                     lm_R2-4-015_door.bc.example.com.
lamp3333._light._udp.bc.example.com    IN TXT
                                     txtver=1;path=/light/right
lamp4444._light._udp.bc.example.com    IN SRV  0 0 5683
                                     lm_R2-4-015_wndw.bc.example.com.
lamp4444._light._udp.bc.example.com    IN TXT
                                     txtver=1;path=/light/left
lamp5555._light._udp.bc.example.com    IN SRV  0 0 5683
                                     lm_R2-4-015_wndw.bc.example.com.
lamp5555._light._udp.bc.example.com    IN TXT
                                     txtver=1;path=/light/middle
lamp6666._light._udp.bc.example.com    IN SRV  0 0 5683
                                     lm_R2-4-015_wndw.bc.example.com.
lamp6666._light._udp.bc.example.com    IN TXT
                                     txtver=1;path=/light/right
pres1234._p-sensor._udp.bc.example.com IN SRV  0 0 5683
                                     ps_R2-4-015_door.bc.example.com.
pres1234._p-sensor._udp.bc.example.com IN TXT
                                     txtver=1;path=/ps

```

To ask for all lamps is equivalent to returning all PTR RR with label `_light.udp.bc.example.com.` from the DNS. When it is required to

filter on the rd=R2-4-015 value in the DNS, additional PTR RRs have to be entered into the DNS.

```
R2-4-015._light._udp.bc.example.com      IN PTR
                                     lamp1111._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com      IN PTR
                                     lamp2222._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com      IN PTR
                                     lamp3333._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com      IN PTR
                                     lamp4444._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com      IN PTR
                                     lamp5555._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com      IN PTR
                                     lamp6666._light._udp.bc.example.com
```

Returning all PTR RRs with label R2-4-015._light._udp.bc.example.com provides all service instances within the domain R2-4-015. This filtering can be handy when there are many rooms. In the example there is only one room, making the filtering superfluous.

The agent can also discover groups that need to be discovered. It queries RD to return all groups which are exported.

```
Req: GET /rd-lookup/gp?exp

Res: 2.05 Content
<coap://[FF05::1]/>;exp;gp="grp_R2-4-015;ins="grp1234";
ep="lm_R2-4-015_wndw";
ep="lm_R2-4-015_door
```

The group with FQDN grp_R2-4-015.bc.example.com can be entered into the DNS by the agent. The accompanying instance name is grp1234. The <ServiceType> is chosen to be _group._udp. The agent enters the following RRs into the DNS.

```
grp_R2-4-015.bc.example.com.      IN AAAA      FF05::1
_group._udp.bc.example.com      IN PTR
                                grp1234._group._udp.bc.example.com
grp1234._group._udp.bc.example.com IN SRV  0 0 5683
                                grp_R2-4-015_door.bc.example.com.
grp1234._group._udp.bc.example.com IN TXT
                                txtver=1;path=/light/grp1
```

12.1.4. RD Operation

The specification of the group can be used by devices other than the luminaries and the sensor to learn the multicast address of the group in a given room. For example a smart phone may be used to adjust the lamps in the room.

After entry into the room, on request of the user, the smart phone queries the presence of RDs and may display all the domain names found on the RDs. The user can, for example, scroll all domains (room names in this case) and select the room that he entered. After selection the phone shows all groups in the selected room with their members. Selecting a group, the user can dim, switch on/off the group of lights, or possibly even create temporary new groups.

In all examples the SLAAC IPv6 address can be exchanged with the FQDN, when a connection to DNS exists. Using the FQDN, a node learns the interface's IPv6 address, or the group's multicast address from DNS. In the same way the presence sensor can learn the multicast address to which it should send its presence messages.

12.2. OMA Lightweight M2M (LWM2M) Example

This example shows how the OMA LWM2M specification makes use of Resource Directory (RD).

OMA LWM2M is a profile for device services based on CoAP, CoRE RD, and other IETF RFCs and drafts. LWM2M defines a simple object model and a number of abstract interfaces and operations for device management and device service enablement.

An LWM2M server is an instance of an LWM2M middleware service layer, containing a Resource Directory along with other LWM2M interfaces defined by the LWM2M specification.

CoRE Resource Directory (RD) is used to provide the LWM2M Registration interface.

LWM2M does not provide for registration domains and does not currently use the rd-group or rd-lookup interfaces.

The LWM2M specification describes a set of interfaces and a resource model used between a LWM2M device and an LWM2M server. Other interfaces, proxies, applications, and function sets are currently out of scope for LWM2M.

The location of the LWM2M Server and RD Function Set is provided by the LWM2M Bootstrap process, so no dynamic discovery of the RD

function set is used. LWM2M Servers and endpoints are not required to implement the `./well-known/core` resource.

12.2.1. The LWM2M Object Model

The OMA LWM2M object model is based on a simple 2 level class hierarchy consisting of Objects and Resources.

An LWM2M Resource is a REST endpoint, allowed to be a single value or an array of values of the same data type.

An LWM2M Object is a resource template and container type that encapsulates a set of related resources. An LWM2M Object represents a specific type of information source; for example, there is a LWM2M Device Management object that represents a network connection, containing resources that represent individual properties like radio signal strength.

Since there may potentially be more than one of a given type object, for example more than one network connection, LWM2M defines instances of objects that contain the resources that represent a specific physical thing.

The URI template for LWM2M consists of a base URI followed by Object, Instance, and Resource IDs:

```
{/base-uri}/{/object-id}/{/object-instance}/{/resource-id}/{/resource-instance}
```

The five variables given here are strings. `base-uri` can also have the special value "undefined" (sometimes called "null" in RFC 6570). Each of the variables `object-instance`, `resource-id`, and `resource-instance` can be the special value "undefined" only if the values behind it in this sequence also are "undefined". As a special case, `object-instance` can be "empty" (which is different from "undefined") if `resource-id` is not "undefined". [_TEMPLATE_TODO]

`base-uri` := Base URI for LWM2M resources or "undefined" for default (empty) base URI

`object-id` := OMNA registered object ID (0-65535)

`object-instance` := Object instance identifier (0-65535) or "undefined"/"empty" (see above) to refer to all instances of an object ID

`resource-id` := OMNA registered resource ID (0-65535) or "undefined" to refer to all resources within an instance

resource-instance := Resource instance identifier or "undefined" to refer to single instance of a resource

LWM2M IDs are 16 bit unsigned integers represented in decimal (no leading zeroes except for the value 0) by URI format strings. For example, a LWM2M URI might be:

/1/0/1

The base uri is empty, the Object ID is 1, the instance ID is 0, the resource ID is 1, and the resource instance is "undefined". This example URI points to internal resource 1, which represents the registration lifetime configured, in instance 0 of a type 1 object (LWM2M Server Object).

12.2.2. LWM2M Register Endpoint

LWM2M defines a registration interface based on the Resource Directory Function Set, described in Section 5. The URI of the LWM2M Resource Directory function set is specified to be "/rd" as recommended in Section 5.2.

LWM2M endpoints register object IDs, for example </1>, to indicate that a particular object type is supported, and register object instances, for example </1/0>, to indicate that a particular instance of that object type exists.

Resources within the LWM2M object instance are not registered with the RD, but may be discovered by reading the resource links from the object instance using GET with a CoAP Content-Format of application/link-format. Resources may also be read as a structured object by performing a GET to the object instance with a Content-Format of senml+json.

When an LWM2M object or instance is registered, this indicates to the LWM2M server that the object and its resources are available for management and service enablement (REST API) operations.

LWM2M endpoints may use the following RD registration parameters as defined in Table 1 :

ep - Endpoint Name
lt - registration lifetime

Endpoint Name is mandatory, all other registration parameters are optional.

Additional optional LWM2M registration parameters are defined:

Name	Query	Validity	Description
Protocol Binding	b	{"U", "UQ", "S", "SQ", "US", "UQS"}	Available Protocols
LWM2M Version	ver	1.0	Spec Version
SMS Number	sms		MSISDN

Table 5: LWM2M Additional Registration Parameters

The following RD registration parameters are not currently specified for use in LWM2M:

et - Endpoint Type
con - Context

The endpoint registration must include a payload containing links to all supported objects and existing object instances, optionally including the appropriate link-format relations.

Here is an example LWM2M registration payload:

```
</1>,</1/0>,</3/0>,</5>
```

This link format payload indicates that object ID 1 (LWM2M Server Object) is supported, with a single instance 0 existing, object ID 3 (LWM2M Device object) is supported, with a single instance 0 existing, and object 5 (LWM2M Firmware Object) is supported, with no existing instances.

12.2.3. Alternate Base URI

If the LWM2M endpoint exposes objects at a base URI other than the default empty base path, the endpoint must register the base URI using `rt="oma.lwm2m"`. An example link payload using alternate base URI would be:

```
</my_lwm2m>;rt="oma.lwm2m",</my_lwm2m/1>,<my_lwm2m/1/0>,<my_lwm2m/5>
```

This link payload indicates that the lwm2m objects will be placed under the base URI `"/my_lwm2m"` and that object ID 1 (server) is supported, with a single instance 0 existing, and object 5 (firmware update) is supported.

12.2.4. LWM2M Update Endpoint Registration

An LWM2M Registration update proceeds as described in Section 5.3, and adds some optional parameter updates:

lt - Registration Lifetime
b - Protocol Binding
sms - MSISDN
link payload - new or modified links

A Registration update is also specified to be used to update the LWM2M server whenever the endpoint's UDP port or IP address are changed.

12.2.5. LWM2M De-Register Endpoint

LWM2M allows for de-registration using the delete method on the returned location from the initial registration operation. LWM2M de-registration proceeds as described in Section 5.4.

13. Acknowledgments

Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Mohit Sethi, Sampo Ukkola and Linyi Tian have provided helpful comments, discussions and ideas to improve and shape this document. Section 9 is based on an earlier draft by Kerry Lynn. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

14. Changelog

changes from -06 to -07

- o added text in the discovery section to allow content format hints to be exposed in the discovery link attributes
- o editorial updates to section 9
- o update author information
- o minor text corrections

Changes from -05 to -06

- o added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

- o added Update Endpoint Links using PATCH
- o http access made explicit in interface specification
- o Added http examples

Changes from -03 to -04:

- o Added http response codes
- o Clarified endpoint name usage
- o Add application/link-format+cbor content-format

Changes from -02 to -03:

- o Added an example for lighting and DNS integration
- o Added an example for RD use in OMA LWM2M
- o Added Read Links operation for link inspection by endpoints
- o Expanded DNS-SD section
- o Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

- o Added a catalogue use case.
- o Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.
- o Additional examples section added for more complex use cases.
- o New DNS-SD mapping section.
- o Added text on endpoint identification and authentication.
- o Error code 4.04 added to Registration Update and Delete requests.
- o Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- o Removed the ETag validation feature.
- o Place holder for the DNS-SD mapping section.
- o Explicitly disabled GET or POST on returned Location.
- o New registry for RD parameters.
- o Added support for the JSON Link Format.
- o Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- o Updated the version and date.

Changes from -04 to -05:

- o Restricted Update to parameter updates.
- o Added pagination support for the Lookup interface.
- o Minor editing, bug fixes and reference updates.
- o Added group support.
- o Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- o Added the ins= parameter back for the DNS-SD mapping.
- o Integrated the Simple Directory Discovery from Carsten.
- o Editorial improvements.
- o Fixed the use of ETags.
- o Fixed tickets 383 and 372

Changes from -02 to -03:

- o Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- o Updated REST interface descriptions to use RFC6570 URI Template format.

- o Introduced an improved RD Lookup design as its own function set.
- o Improved the security considerations section.
- o Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

- o Added a terminology section.
- o Changed the inclusion of an ETag in registration or update to a MAY.
- o Added the concept of an RD Domain and a registration parameter for it.
- o Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- o Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

15. References

15.1. Normative References

- [I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-04 (work in progress), November 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<http://www.rfc-editor.org/info/rfc6335>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<http://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<http://www.rfc-editor.org/info/rfc6763>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <<http://www.rfc-editor.org/info/rfc7396>>.

15.2. Informative References

- [I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., and M. Koster, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-04 (work in progress), October 2015.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<http://www.rfc-editor.org/info/rfc1034>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<http://www.rfc-editor.org/info/rfc1123>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<http://www.rfc-editor.org/info/rfc3629>>.

- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <<http://www.rfc-editor.org/info/rfc5198>>.
- [RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <<http://www.rfc-editor.org/info/rfc6775>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<http://www.rfc-editor.org/info/rfc7390>>.

Editorial Comments

[_TEMPLATE_TODO] This text needs some help from an RFC 6570 expert.

Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Phone: +1-707-502-5136
Email: Michael.Koster@smarththings.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

CoRE
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2021

C. Amsüss, Ed.
Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
7 March 2021

CoRE Resource Directory
draft-ietf-core-resource-directory-28

Abstract

In many IoT applications, direct discovery of resources is not practical due to sleeping nodes, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources. The input to an RD is composed of links and the output is composed of links constructed from the information stored in the RD. This document specifies the web interfaces that an RD supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with an RD are defined.

Note to Readers

Discussion of this document takes place on the CORE Working Group mailing list (core@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/browse/core/> (<https://mailarchive.ietf.org/arch/browse/core/>).

Source for this draft and an issue tracker can be found at <https://github.com/core-wg/resource-directory> (<https://github.com/core-wg/resource-directory>).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 September 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
2. Terminology	4
3. Architecture and Use Cases	6
3.1. Principles	6
3.2. Architecture	7
3.3. RD Content Model	8
3.4. Link-local addresses and zone identifiers	12
3.5. Use Case: Cellular M2M	12
3.6. Use Case: Home and Building Automation	13
3.7. Use Case: Link Catalogues	14
4. RD discovery and other interface-independent components	14
4.1. Finding a Resource Directory	15
4.1.1. Resource Directory Address Option (RDAO)	17
4.1.2. Using DNS-SD to discover a Resource Directory	19
4.2. Payload Content Formats	19
4.3. URI Discovery	19
5. Registration	22
5.1. Simple Registration	27
5.2. Third-party registration	29
5.3. Operations on the Registration Resource	30

5.3.1.	Registration Update	30
5.3.2.	Registration Removal	34
5.3.3.	Further operations	34
5.3.4.	Request freshness	35
6.	RD Lookup	37
6.1.	Resource lookup	37
6.2.	Lookup filtering	38
6.3.	Resource lookup examples	40
6.4.	Endpoint lookup	42
7.	Security policies	43
7.1.	Endpoint name	44
7.1.1.	Random endpoint names	44
7.2.	Entered resources	44
7.3.	Link confidentiality	45
7.4.	Segmentation	46
7.5.	First-Come-First-Remembered: A default policy	46
8.	Security Considerations	48
8.1.	Discovery	48
8.2.	Endpoint Identification and Authentication	48
8.3.	Access Control	49
8.4.	Denial of Service Attacks	49
8.5.	Skipping freshness checks	50
9.	IANA Considerations	50
9.1.	Resource Types	50
9.2.	IPv6 ND Resource Directory Address Option	51
9.3.	RD Parameter Registry	51
9.3.1.	Full description of the "Endpoint Type" RD Parameter	54
9.4.	"Endpoint Type" (et=) RD Parameter values	54
9.5.	Multicast Address Registration	55
9.6.	Well-Known URIs	55
9.7.	Service Names and Transport Protocol Port Number Registry	55
10.	Examples	56
10.1.	Lighting Installation	56
10.1.1.	Installation Characteristics	56
10.1.2.	RD entries	57
10.2.	OMA Lightweight M2M (LwM2M)	60
11.	Acknowledgments	61
12.	Changelog	61
13.	References	76
13.1.	Normative References	76
13.2.	Informative References	77
Appendix A.	Groups Registration and Lookup	80
Appendix B.	Web links and the Resource Directory	82
B.1.	A simple example	82
B.1.1.	Resolving the URIs	82
B.1.2.	Interpreting attributes and relations	83

B.2. A slightly more complex example	83
B.3. Enter the Resource Directory	84
B.4. A note on differences between link-format and Link header fields	86
Appendix C. Limited Link Format	86
Authors' Addresses	87

1. Introduction

In the work on Constrained RESTful Environments (CoRE), a REST architecture suitable for constrained nodes (e.g. with limited RAM and ROM [RFC7228]) and networks (e.g. 6LoWPAN [RFC4944]) has been established and is used in Internet-of-Things (IoT) or machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC8288]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. However, [RFC6690] only describes how to discover resources from the web server that hosts them by querying `"/.well-known/core"`. In many constrained scenarios, direct discovery of resources is not practical due to sleeping nodes, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that an RD supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with an RD are defined. Although the examples in this document show the use of these interfaces with CoAP [RFC7252], they can be applied in an equivalent manner to HTTP [RFC7230].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The term "byte" is used in its now customary sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC3986], [RFC8288] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

resolve against

The expression "a URI-reference is resolved against a base URI" is used to describe the process of [RFC3986] Section 5.2.

Noteworthy corner cases are that if the URI-reference is a (full) URI and resolved against any base URI, that gives the original full URI, and that resolving an empty URI reference gives the base URI without any fragment identifier.

Resource Directory (RD)

A web entity that stores information about web resources and implements the REST interfaces defined in this specification for discovery, for the creation, maintenance and removal of registrations, and for lookup of the registered resources.

Sector

In the context of an RD, a sector is a logical grouping of endpoints.

The abbreviation "d=" is used for the sector in query parameters for compatibility with deployed implementations.

Endpoint

Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the RD. An endpoint is identified by its endpoint name, which is included during registration, and has a unique name within the associated sector of the registration.

Registration Base URI

The Base URI of a Registration is a URI that typically gives scheme and authority information about an Endpoint. The Registration Base URI is provided at registration time, and is used by the RD to resolve relative references of the registration into URIs.

Target

The target of a link is the destination address (URI) of the link. It is sometimes identified with "href=", or displayed as "<target>". Relative targets need resolving with respect to the Base URI (section 5.2 of [RFC3986]).

This use of the term Target is consistent with [RFC8288]'s use of the term.

Context

The context of a link is the source address (URI) of the link, and describes which resource is linked to the target. A link's context is made explicit in serialized links as the "anchor=" attribute.

This use of the term Context is consistent with [RFC8288]'s use of the term.

Directory Resource

A resource in the RD containing registration resources.

Registration Resource

A resource in the RD that contains information about an Endpoint and its links.

Commissioning Tool

Commissioning Tool (CT) is a device that assists during installation events by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

Registrant-ep

Registrant-ep is the endpoint that is registered into the RD. The registrant-ep can register itself, or a CT registers the registrant-ep.

RDAO

Resource Directory Address Option. A new IPv6 Neighbor Discovery option defined for announcing an RD's address.

3. Architecture and Use Cases

3.1. Principles

The RD is primarily a tool to make discovery operations more efficient than querying /.well-known/core on all connected devices, or across boundaries that would limit those operations.

It provides information about resources hosted by other devices that could otherwise only be obtained by directly querying the /.well-known/core resource on these other devices, either by a unicast request or a multicast request.

Information SHOULD only be stored in the RD if it can be obtained by querying the described device's /.well-known/core resource directly.

Data in the RD can only be provided by the device which hosts those data or a dedicated Commissioning Tool (CT). These CTs act on behalf of endpoints too constrained, or generally unable, to present that information themselves. No other client can modify data in the RD. Changes to the information in the RD do not propagate automatically back to the web servers from where the information originated.

3.2. Architecture

The RD architecture is illustrated in Figure 1. An RD is used as a repository of registrations describing resources hosted on other web servers, also called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port. A physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain RD registrations, and for endpoints to lookup resources from the RD. An RD can be logically segmented by the use of Sectors.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Registrations in the RD are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a registration. It is also possible for an RD to fetch Web Links from endpoints and add their contents to its registrations.

At the first registration of an endpoint, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registrations.

A lookup interface for discovering any of the Web Links stored in the RD is provided using the CoRE Link Format.

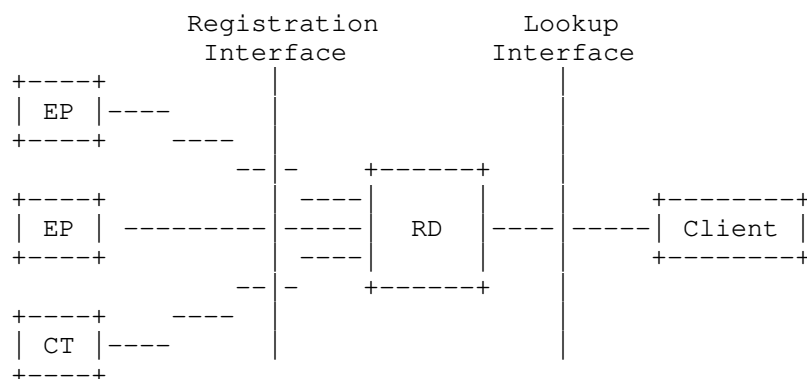


Figure 1: The RD architecture.

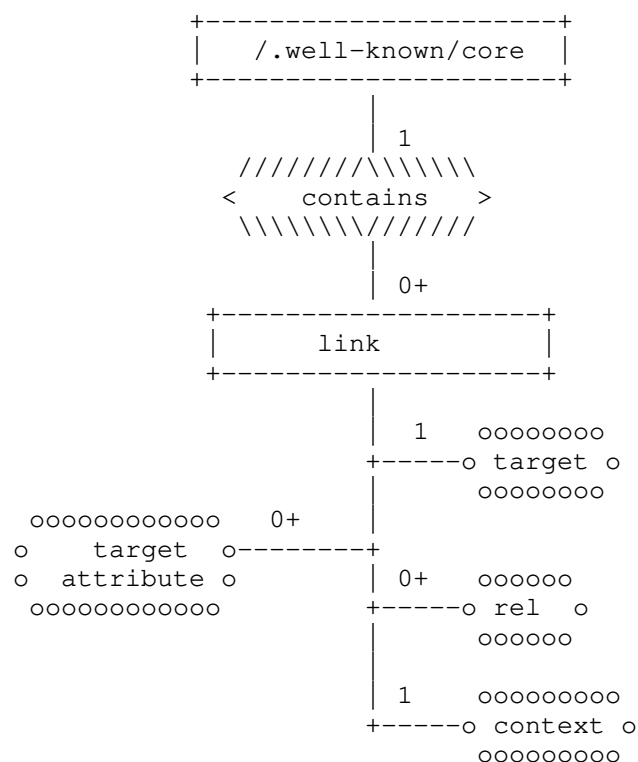
A Registrant-EP MAY keep concurrent registrations to more than one RD at the same time if explicitly configured to do so, but that is not expected to be supported by typical EP implementations. Any such registrations are independent of each other. The usual expectation when multiple discovery mechanisms or addresses are configured is that they constitute a fall-back path for a single registration.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 2 and Figure 3 model the contents of `/.well-known/core` and the RD respectively, with entity-relationship diagrams [ER]. Entities (rectangles) are used for concepts that exist independently. Attributes (ovals) are used for concepts that exist only in connection with a related entity. Relations (diamonds) give a semantic meaning to the relation between entities. Numbers specify the cardinality of the relations.

Some of the attribute values are URIs. Those values are always full URIs and never relative references in the information model. They can, however, be expressed as relative references in serializations, and often are.

These models provide an abstract view of the information expressed in link-format documents and an RD. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

Figure 2: ER Model of the content of `/.well-known/core`

The model shown in Figure 2 models the contents of `/.well-known/core` which contains:

- * a set of links belonging to the hosting web server

The web server is free to choose links it deems appropriate to be exposed in its `/.well-known/core`. Typically, the links describe resources that are served by the host, but the set can also contain links to resources on other servers (see examples in [RFC6690] page 14). The set does not necessarily contain links to all resources served by the host.

A link has the following attributes (see [RFC8288]):

- * **Zero or more link relations:** They describe relations between the link context and the link target.

In link-format serialization, they are expressed as space-separated values in the `rel` attribute, and default to `hosts`.

- * A link context URI: It defines the source of the relation, e.g. `_who_ "hosts" something`.

In link-format serialization, it is expressed in the "anchor" attribute and defaults to the Origin of the target (practically: the target with its path and later components removed)

- * A link target URI: It defines the destination of the relation (e.g. `_what_ is hosted`), and is the topic of all target attributes.

In link-format serialization, it is expressed between angular brackets, and sometimes called the "href".

- * Other target attributes (e.g. resource type (rt), interface (if), or content format (ct)). These provide additional information about the target URI.

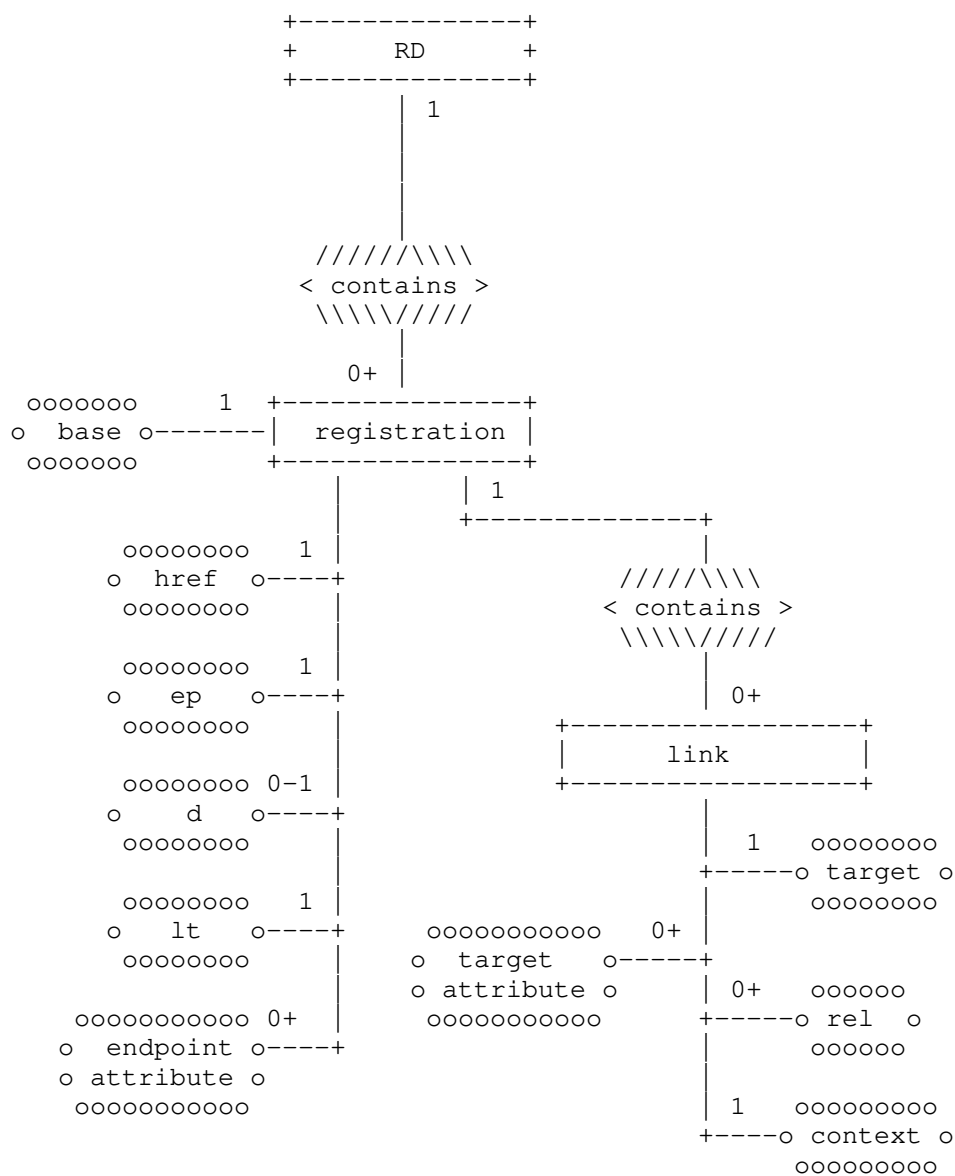


Figure 3: ER Model of the content of the RD

The model shown in Figure 3 models the contents of the RD which contains in addition to /.well-known/core:

* 0 to n Registrations of endpoints,

A registration is associated with one endpoint. A registration defines a set of links as defined for `/.well-known/core`. A Registration has six types of attributes:

- * an endpoint name ("ep", a Unicode string) unique within a sector
- * a Registration Base URI ("base", a URI typically describing the `scheme://authority` part)
- * a lifetime ("lt"),
- * a registration resource location inside the RD ("href"),
- * optionally a sector ("d", a Unicode string)
- * optional additional endpoint attributes (from Section 9.3)

The cardinality of "base" is currently 1; future documents are invited to extend the RD specification to support multiple values (e.g. `[I-D.silverajan-core-coap-protocol-negotiation]`). Its value is used as a Base URI when resolving URIs in the links contained in the endpoint.

Links are modelled as they are in Figure 2.

3.4. Link-local addresses and zone identifiers

Registration Base URIs can contain link-local IP addresses. To be usable across hosts, those cannot be serialized to contain zone identifiers (see [RFC6874] Section 1).

Link-local addresses can only be used on a single link (therefore RD servers cannot announce them when queried on a different link), and lookup clients using them need to keep track of which interface they got them from.

Therefore, it is advisable in many scenarios to use addresses with larger scope if available.

3.5. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded wireless interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. The ambition in such systems is to build them from reusable components. These speed up

development and deployment, and enable shared use of machines across different applications. One crucial component of such systems is the discovery of resources (and thus the endpoints they are hosted on) capable of providing required information at a given time or acting on instructions from the end users.

Imagine a scenario where endpoints installed on vehicles enable tracking of the position of these vehicles for fleet management purposes and allow monitoring of environment parameters. During the boot-up process endpoints register with an RD, which is hosted by the mobile operator or somewhere in the cloud. Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web applications for environment monitoring contact the RD, look up the endpoints capable of providing information about the environment using an appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server), and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look up for EPs deployed on the vehicles the application is responsible for.

3.6. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of IoT web services. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices. Both can use the common RD infrastructure to establish device interactions efficiently, but can pick security policies suitable for their needs.

Two phases can be discerned for a network servicing the system: (1) installation and (2) operation. During the operational phase, the network is connected to the Internet with a Border Router (e.g. a 6LoWPAN Border Router (6LBR), see [RFC6775]) and the nodes connected to the network can use the Internet services that are provided by the Internet Provider or the network administrator. During the installation phase, the network is completely stand-alone, no Border Router is connected, and the network only supports the IP communication between the connected nodes. The installation phase is

usually followed by the operational phase. As an RD's operations work without hard dependencies on names or addresses, it can be used for discovery across both phases.

3.7. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. An RD can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to an RD. Applications wishing to consume the data can use RD Lookup to discover and resolve links to the desired resources and endpoints. The RD service need not be coupled with the data intermediary service. Mapping of RDs to data intermediaries may be many-to-many.

Metadata in web link formats like [RFC6690] which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links, need to be supported by RDs. External catalogues that are represented in other formats may be converted to common web linking formats for storage and access by RDs. Since it is common practice for these to be encoded in URNs [RFC8141], simple and lossless structural transforms should generally be sufficient to store external metadata in RDs.

The additional features of an RD allow sectors to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Application groups with multicast addresses may be defined to support efficient data transport.

4. RD discovery and other interface-independent components

This and the following sections define the required set of REST interfaces between an RD, endpoints and lookup clients. Although the examples throughout these sections assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. The multicast discovery and simple registration operations are exceptions to that, as they rely on mechanisms unavailable in HTTP. In all definitions in these sections, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces.

All operations on the contents of the RD MUST be atomic and idempotent.

For several operations, interface templates are given in list form; those describe the operation participants, request codes, URIs, content formats and outcomes. Sections of those templates contain normative content about Interaction, Method, URI Template and URI Template Variables as well as the details of the Success condition. The additional sections on options like Content-Format and on Failure codes give typical cases that an implementation of the RD should deal with. Those serve to illustrate the typical responses to readers who are not yet familiar with all the details of CoAP based interfaces; they do not limit what a server may respond under atypical circumstances.

REST clients (registrant-EPs and CTs during registration and maintenance, lookup clients, RD servers during simple registrations) must be prepared to receive any unsuccessful code and act upon it according to its definition, options and/or payload to the best of their capabilities, falling back to failing the operation if recovery is not possible. In particular, they SHOULD retry the request upon 5.03 (Service Unavailable; 503 in HTTP) according to the Max-Age (Retry-After in HTTP) option, and SHOULD fall back to link-format when receiving 4.15 (Unsupported Content-Format; 415 in HTTP).

An RD MAY make the information submitted to it available to further directories (subject to security policies on link confidentiality), if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

4.1. Finding a Resource Directory

A (re-)starting device may want to find one or more RDs before it can discover their URIs. Dependent on the operational conditions, one or more of the techniques below apply.

The device may be pre-configured to exercise specific mechanisms for finding the RD:

1. It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an appropriate RD. (Instead of using an anycast address, a multicast address can also be preconfigured. The RD servers then need to configure one of their interfaces with this multicast address.)
2. It may be configured with a DNS name for the RD and use DNS to return the IP address of the RD; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.
3. It may be configured to use a service discovery mechanism such as DNS-SD, as outlined in Section 4.1.2.

For cases where the device is not specifically configured with a way to find an RD, the network may want to provide a suitable default.

1. The IPv6 Neighbor Discovery option RDAO Section 4.1.1 can do that.
2. When DHCP is in use, this could be provided via a DHCP option (no such option is defined at the time of writing).

Finally, if neither the device nor the network offers any specific configuration, the device may want to employ heuristics to find a suitable RD.

The present specification does not fully define these heuristics, but suggests a number of candidates:

1. In a 6LoWPAN, just assume the Border Router (6LBR) can act as an RD (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".
2. In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[MCD1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

When answering a multicast request directed at a link-local group, the RD may want to respond from a routable address; this makes it easier for registrants to use one of their own routable addresses for

registration. When [RFC6724] is used for source address selection, this can be achieved by applying the changes of its Section 10.4, picking public addresses in its Section 5 Rule 7, and superseding rule 8 with preferring the source address's precedence.

As some of the RD addresses obtained by the methods listed here are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

The following RD discovery mechanisms are recommended:

- * In managed networks with border routers that need stand-alone operation, the RDAO option is recommended (e.g. operational phase described in Section 3.6).
- * In managed networks without border router (no Internet services available), the use of a preconfigured anycast address is recommended (e.g. installation phase described in Section 3.6).
- * In networks managed using DNS-SD, the use of DNS-SD for discovery as described in Section 4.1.2 is recommended.

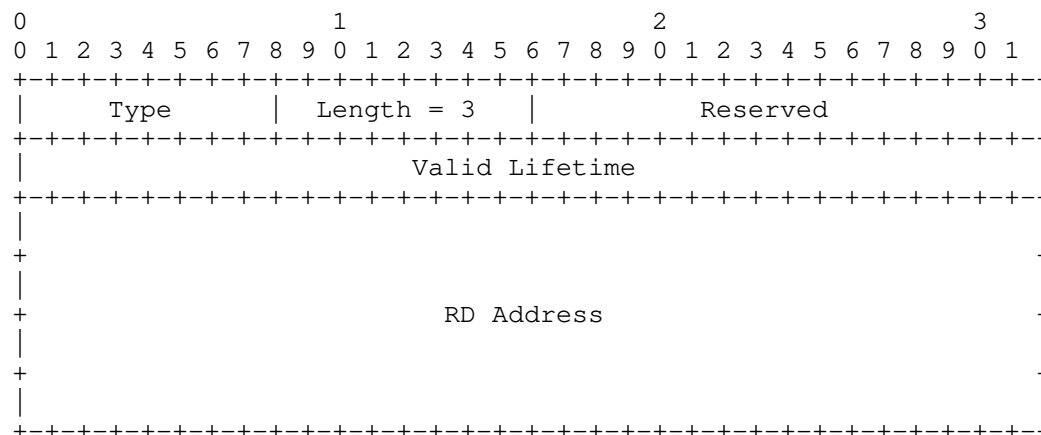
The use of multicast discovery in mesh networks is NOT RECOMMENDED.

4.1.1. Resource Directory Address Option (RDAO)

The Resource Directory Address Option (RDAO) carries information about the address of the RD in RAs (Router Advertisements) of IPv6 Neighbor Discovery (ND), similar to how RDNSS options [RFC8106] are sent. This information is needed when endpoints cannot discover the RD with a link-local or realm-local scope multicast address, for instance because the endpoint and the RD are separated by a Border Router (6LBR). In many circumstances the availability of DHCP cannot be guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many RD addresses.

The RDAO format is:



Fields:

Type: TBD38

Length: 8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3.

```
Reserved:      This field is unused.  It MUST be
                initialized to zero by the sender and
                MUST be ignored by the receiver.
```

Valid Lifetime: 32-bit unsigned integer. The length of time in seconds (relative to the time the packet is received) that this RD address is valid. A value of all zero bits (0x0) indicates that this RD address is not valid anymore.

RD Address: IPv6 address of the RD.

Figure 4: Resource Directory Address Option

4.1.2. Using DNS-SD to discover a Resource Directory

An RD can advertise its presence in DNS-SD [RFC6763] using the service name "_core-rd._udp" (for CoAP), "_core-rd-dtls._udp" (for CoAP over DTLS), "_core-rd._tcp" (for CoAP over TCP) or "_core-rd-tls._tcp" (for CoAP over TLS) defined in this document. (For the WebSocket transports of CoAP, no service is defined as DNS-SD is typically unavailable in environments where CoAP over WebSockets is used).

The selection of the service indicates the protocol used, and the SRV record points the client to a host name and port to use as a starting point for the URI discovery steps of Section 4.3.

This section is a simplified concrete application of the more generic mechanism specified in [I-D.ietf-core-rd-dns-sd].

4.2. Payload Content Formats

RDs implementing this specification MUST support the application/link-format content format (ct=40).

RDs implementing this specification MAY support additional content formats.

Any additional content format supported by an RD implementing this specification SHOULD be able to express all the information expressible in link-format. It MAY be able to express information that is inexpressible in link-format, but those expressions SHOULD be avoided where possible.

4.3. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690] after having discovered a host as described in Section 4.1.

Discovery of the RD registration URI is performed by sending either a multicast or unicast GET request to `"/.well-known/core"` and including a Resource Type (rt) parameter [RFC6690] with the value `"core.rd"` in the query string. Likewise, a Resource Type parameter value of `"core.rd-lookup"` is used to discover the URIs for RD Lookup operations, `core.rd*` is used to discover all URIs for RD operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI of the RD function returned and the corresponding Resource Type. When

performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network (see Section 9.5).

An RD MAY provide hints about the content-formats it supports in the links it exposes or registers, using the "ct" target attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the RD.

HTTP does not support multicast and consequently only unicast discovery can be supported at the using the HTTP `"/.well-known/core"` resource.

RDs implementing this specification MUST support query filtering for the `rt` parameter as defined in [RFC6690].

While the link targets in this discovery step are often expressed in path-absolute form, this is not a requirement. Clients of the RD SHOULD therefore accept URIs of all schemes they support, both as URIs and relative references, and not limit the set of discovered URIs to those hosted at the address used for URI discovery.

With security policies where the client requires the RD to be authorized to act as an RD, that authorization may be limited to resources on which the authorized RD advertises the adequate resource types. Clients that have obtained links they can not rely on yet can repeat the URI discovery step at the `/.well-known/core` resource of the indicated host to obtain the resource type information from an authorized source.

The URI Discovery operation can yield multiple URIs of a given resource type. The client of the RD can use any of the discovered addresses initially.

The discovery request interface is specified as follows (this is exactly the Well-Known Interface of [RFC6690] Section 4, with the additional requirement that the server MUST support query filtering):

Interaction: EP, CT or Client -> RD

Method: GET

URI Template: `/.well-known/core{?rt}`

URI Template Variables: `rt` := Resource Type. SHOULD contain one of the values `"core.rd"`, `"core.rd-lookup*"`, `"core.rd-lookup-res"`, `"core.rd-lookup-ep"`, or `"core.rd*"`

Accept: absent, application/link-format or any other media type representing web links

The following response is expected on this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format or other web link payload containing one or more matching entries for the RD resource.

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource location, in this example, is /rd, and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that it is up to the RD to choose its RD locations.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content

Payload:

```
</rd>;rt=core.rd;ct=40,  
</rd-lookup/ep>;rt=core.rd-lookup-ep;ct=40,  
</rd-lookup/res>;rt=core.rd-lookup-res;ct=40
```

Figure 5: Example discovery exchange

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as a CBOR and JSON representation from [I-D.ietf-core-links-json] (which have no numeric values assigned yet, so they are shown as TBD64 and TBD504 as in that draft). The RD resource locations /rd, and /rd-lookup are example values. The server in this example also indicates that it is capable of providing observation on resource lookups.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content

Payload:

```
</rd>;rt=core.rd;ct="40 65225",  
</rd-lookup/res>;rt=core.rd-lookup-res;ct="40 TBD64 TBD504";obs,  
</rd-lookup/ep>;rt=core.rd-lookup-ep;ct="40 TBD64 TBD504"
```

Figure 6: Example discovery exchange indicating additional content-formats

For maintenance, management and debugging, it can be useful to identify the components that constitute the RD server. The identification can be used to find client-server incompatibilities, supported features, required updates and other aspects. The Well-Known interface described in Section 4 of [RFC6690] can be used to find such data.

It would typically be stored in an implementation information link (as described in [I-D.bormann-t2trg-rel-impl]):

```
Req: GET /.well-known/core?rel=impl-info
```

```
Res: 2.05 Content
```

```
Payload:
```

```
<http://software.example.com/shiny-resource-directory/1.0beta1>;  
rel=impl-info
```

Figure 7: Example exchange of obtaining implementation information, using the relation type currently proposed in the work-in-progress document

Note that depending on the particular server's architecture, such a link could be anchored at the RD server's root (as in this example), or at individual RD components. The latter is to be expected when different applications are run on the same server.

5. Registration

After discovering the location of an RD, a registrant-ep or CT MAY register the resources of the registrant-ep using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690] or other representations of web links, along with query parameters indicating the name of the endpoint, and optionally the sector, lifetime and base URI of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. The receiving endpoint MUST use that location when refreshing registrations using this interface. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The creating endpoint is responsible for refreshing the registration resource within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d (sector) does not create multiple registration resources.

The following rules apply for a registration request targeting a given (ep, d) value pair:

- * When the (ep, d) value pair of the registration-request is different from any existing registration, a new registration is generated.
- * When the (ep, d) value pair of the registration-request is equal to an existing registration, the content and parameters of the existing registration are replaced with the content of the registration request. Like the later changes to registration resources, security policies (Section 7) usually require such requests to come from the same device.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. They are resolved against the base URI of the registration, which is provided either explicitly in the "base" parameter or constructed implicitly from the requester's URI as constructed from its network address and scheme.

For media types to which Appendix C applies (i.e. documents in application/link-format), request bodies MUST be expressed in Limited Link Format.

The registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+rd}{?ep,d,lt,base,extra-attrs*}

URI Template Variables: rd := RD registration URI (mandatory).
This is the location of the RD, as obtained from discovery.

ep := Endpoint name (mostly mandatory).
The endpoint name is an identifier that MUST be unique within a sector.

As the endpoint name is a Unicode string, it is encoded in UTF-8 (and possibly pct-encoded) during variable expansion (see [RFC6570] Section 3.2.1). The endpoint name MUST NOT contain any character in the inclusive ranges 0-31 or 127-159.

The maximum length of this parameter is 63 UTF-8 encoded bytes.

If the RD is configured to recognize the endpoint to be authorized to use exactly one endpoint name, the RD assigns that name. In that case, giving the endpoint name becomes optional for the client; if the client gives any other endpoint name, it is not authorized to perform the registration.

`d` := Sector (optional). The sector to which this endpoint belongs. When this parameter is not present, the RD MAY associate the endpoint with a configured default sector (possibly based on the endpoint's authorization) or leave it empty.

The sector is encoded like the `ep` parameter, and is limited to 63 UTF-8 encoded bytes as well.

`lt` := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included in the initial registration, a default value of 90000 (25 hours) SHOULD be assumed.

`base` := Base URI (optional). This parameter sets the base URI of the registration, under which the relative links in the payload are to be interpreted. The specified URI typically does not have a path component of its own, and MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore usually of the shape "scheme://authority" for HTTP and CoAP URIs. The URI SHOULD NOT have a query or fragment component as any non-empty relative part in a reference would remove those parts from the resulting URI.

In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. The Base URI is consecutively constructed by concatenating the used protocol's scheme with the characters "://", the requester's source address as an address literal and ":" followed by its port (if it was not the protocol's default one) in analogy to [RFC7252] Section 6.5.

This parameter is mandatory when the directory is filled by a third party such as an commissioning tool.

If the registrant-ep uses an ephemeral port to register with, it MUST include the base parameter in the registration to provide a valid network path.

A registrant that cannot be reached by potential lookup clients

at the address it registers from (e.g. because it is behind some form of Network Address Translation (NAT)) MUST provide a reachable base address with its registration.

If the Base URI contains a link-local IP literal, it MUST NOT contain a Zone Identifier, and MUST be local to the link on which the registration request is received.

Endpoints that register with a base that contains a path component cannot efficiently express their registrations in Limited Link Format (Appendix C). Those applications should use different representations of links to which Appendix C is not applicable (e.g. [I-D.hartke-t2trg-coral]).

extra-attrs := Additional registration attributes (optional). The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

Content-Format: application/link-format or any other indicated media type representing web links

The following response is expected on this interface:

Success: 2.01 "Created" or 201 "Created". The Location-Path option or Location header field MUST be included in the response. This location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links.

A registration with an already registered ep and d value pair responds with the same success code and location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

The location MUST NOT have a query or fragment component, as that could conflict with query parameters during the Registration Update operation. Therefore, the Location-Query option MUST NOT be present in a successful response.

If the registration fails, including request timeouts, or if delays from Service Unavailable responses with Max-Age or Retry-After accumulate to exceed the registrant's configured timeouts, it SHOULD

pick another registration URI from the "URI Discovery" step and if there is only one or the list is exhausted, pick other choices from the "Finding a Resource Directory" step. Care has to be taken to consider the freshness of results obtained earlier, e.g. of the result of a `"/.well-known/core"` response, the lifetime of an RDAO option and of DNS responses. Any rate limits and persistent errors from the "Finding a Resource Directory" step must be considered for the whole registration time, not only for a single operation.

The following example shows a registrant-ep with the name "node1" registering two resources to an RD using this interface. The location `"/rd"` is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel=describedby

Res: 2.01 Created
Location-Path: /rd/4521
```

Figure 8: Example registration payload

An RD may optionally support HTTP. Here is an example of almost the same registration operation above, when done using HTTP.

```
Req:
POST /rd?ep=node1&base=http://[2001:db8:1::1] HTTP/1.1
Host: rd.example.com
Content-Type: application/link-format

</sensors/temp>;rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel=describedby

Res:
HTTP/1.1 201 Created
Location: /rd/4521
```

Figure 9: Example registration payload as expressed using HTTP

5.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to an RD as described in Section 5. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification **MUST** implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the registrant-ep makes available the hosted resources that it wants to be discovered, as links on its `"/.well-known/core"` interface as specified in [RFC6690]. The links in that document are subject to the same limitations as the payload of a registration (with respect to Appendix C).

- * The registrant-ep finds one or more addresses of the directory server as described in Section 4.1.
- * The registrant-ep sends (and regularly refreshes with) a POST request to the `"/.well-known/rd"` URI of the directory server of choice. The body of the POST request is empty, and triggers the resource directory server to perform GET requests at the requesting registrant-ep's `/.well-known/core` to obtain the link-format payload to register.

The registrant-ep includes the same registration parameters in the POST request as it would with a regular registration per Section 5. The registration base URI of the registration is taken from the registrant-ep's network address (as is default with regular registrations).

Example request from registrant-EP to RD (unanswered until the next step):

```
Req: POST /.well-known/rd?lt=6000&ep=node1
(No payload)
```

Figure 10: First half example exchange of a simple registration

- * The RD queries the registrant-ep's discovery resource to determine the success of the operation. It **SHOULD** keep a cache of the discovery resource and not query it again as long as it is fresh.

Example request from the RD to the registrant-EP:

```
Req: GET /.well-known/core
Accept: 40
```

```
Res: 2.05 Content
Content-Format: 40
Payload:
</sen/temp>
```

Figure 11: Example exchange of the RD querying the simple endpoint

With this response, the RD would answer the previous step's request:

```
Res: 2.04 Changed
```

Figure 12: Second half example exchange of a simple registration

The sequence of fetching the registration content before sending a successful response was chosen to make responses reliable, and the point about caching was chosen to still allow very constrained registrants. Registrants **MUST** be able to serve a GET request to `"/.well-known/core"` after having requested registration. Constrained devices **MAY** regard the initial request as temporarily failed when they need RAM occupied by their own request to serve the RD's GET, and retry later when the RD already has a cached representation of their discovery resources. Then, the RD can reply immediately and the registrant can receive the response.

The simple registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: `/.well-known/rd{?ep,d,lt,extra-attrs*}`

URI Template Variables are as they are for registration in Section 5. The base attribute is not accepted to keep the registration interface simple; that rules out registration over CoAP-over-TCP or HTTP that would need to specify one. For some time during this document's development, the URI template `"/.well-known/core{?ep,...}"` has been in use instead.

The following response is expected on this interface:

```
Success: 2.04 "Changed".
```

For the second interaction triggered by the above, the registrant-ep takes the role of server and the RD the role of client. (Note that this is exactly the Well-Known Interface of [RFC6690] Section 4):

Interaction: RD -> EP

Method: GET

URI Template: /.well-known/core

The following response is expected on this interface:

Success: 2.05 "Content".

When the RD uses any authorization credentials to access the endpoint's discovery resource, or when it is deployed in a location where third parties might reach it but not the endpoint, it SHOULD verify that the apparent registrant-ep intends to register with the given registration parameters before revealing the obtained discovery information to lookup clients. An easy way to do that is to verify the simple registration request's sender address using the Echo option as described in [I-D.ietf-core-echo-request-tag] Section 2.4.

The RD MUST delete registrations created by simple registration after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

5.2. Third-party registration

For some applications, even Simple Registration may be too taxing for some very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the RD can be filled by a third party device, called a Commissioning Tool (CT). The commissioning tool can fill the RD from a database or other means. For that purpose scheme, IP address and port of the URI of the registered device is the value of the "base" parameter of the registration described in Section 5.

It should be noted that the value of the "base" parameter applies to all the links of the registration and has consequences for the anchor value of the individual links as exemplified in Appendix B. An eventual (currently non-existing) "base" attribute of the link is not affected by the value of "base" parameter in the registration.

5.3. Operations on the Registration Resource

This section describes how the registering endpoint can maintain the registrations that it created. The registering endpoint can be the registrant-ep or the CT. The registrations are resources of the RD.

An endpoint should not use this interface for registrations that it did not create. This is usually enforced by security policies, which in general require equivalent credentials for creation of and operations on a registration.

After the initial registration, the registering endpoint retains the returned location of the registration resource for further operations, including refreshing the registration in order to extend the lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this endpoint. The RD SHOULD continue to provide access to the registration resource after a registration time-out occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of garbage collection. If the registration resource is removed, the corresponding endpoint will need to be re-registered.

The registration resource may also be used cancel the registration using DELETE, and to perform further operations beyond the scope of this specification.

Operations on the registration resource are sensitive to reordering; Section 5.3.4 describes how order is restored.

The operations on the registration resource are described below.

5.3.1. Registration Update

The update interface is used by the registering endpoint to refresh or update its registration with an RD. To use the interface, the registering endpoint sends a POST request to the registration resource returned by the initial registration operation.

An update MAY update registration parameters like lifetime, base URI or others. Parameters that are not being changed should not be included in an update. Adding parameters that have not changed increases the size of the message but does not have any other implications. Parameters are included as query parameters in an update operation as in Section 5.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the base URI of the registration is changed in an update, relative references submitted in the original registration or later updates are resolved anew against the new base.

The registration update operation only describes the use of POST with an empty payload. Future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration (see Section 5.3.3).

The update registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+location}{?lt,base,extra-attrs*}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 90000) SHOULD be used.

base := Base URI (optional). This parameter updates the Base URI established in the original registration to a new value, and is subject to the same restrictions as in the registration.

If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the relative links present in the payload of the original registration.

If the parameter is not set in the request but was set before, the previous Base URI value is kept unmodified.

If the parameter is not set in the request and was not set before either, the source address and source port of the update request are stored as the Base URI.

extra-attrs := Additional registration

attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Note that this default behavior does not allow removing an endpoint attribute in an update. For attributes whose functionality depends on the endpoints' ability to remove them in an update, it can make sense to define a value whose presence is equivalent to the absence of a value. As an alternative, an extension can define different updating rules for their attributes. That necessitates either discovery of whether the RD is aware of that extension, or tolerating the default behavior.

Content-Format: none (no payload)

The following responses are expected on this interface:

Success: 2.04 "Changed" or 204 "No Content" if the update was successfully processed.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have been removed).

If the registration update fails in any way, including "Not Found" and request timeouts, or if the time indicated in a Service Unavailable Max-Age/Retry-After exceeds the remaining lifetime, the registering endpoint SHOULD attempt registration again.

The following example shows how the registering endpoint resets the timeout on its registration resource at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

Figure 13: Example update of a registration

The following example shows the registering endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the registering endpoint set the following values:

* endpoint name (ep)=endpoint1

* lifetime (lt)=500

* Base URI (base)=coap://local-proxy-old.example.com

* payload of Figure 8

The initial state of the RD is reflected in the following request:

Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content

Payload:

```
<coap://local-proxy-old.example.com/sensors/temp>;
  rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="coap://local-proxy-old.example.com/sensors/temp";
  rel=describedby
```

Figure 14: Example lookup before a change to the base address

The following example shows the registering endpoint changing the Base URI to "coaps://new.example.com:5684":

Req: POST /rd/4521?base=coaps://new.example.com

Res: 2.04 Changed

Figure 15: Example registration update that changes the base address

The consecutive query returns:

Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content

Payload:

```
<coaps://new.example.com/sensors/temp>;
  rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="coaps://new.example.com/sensors/temp";
  rel=describedby
```

Figure 16: Example lookup after a change to the base address

5.3.2. Registration Removal

Although RD registrations have soft state and will eventually timeout after their lifetime, the registering endpoint SHOULD explicitly remove an entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP or CT -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

The following responses are expected on this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may already have been removed).

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

Figure 17: Example of a registration removal

5.3.3. Further operations

Additional operations on the registration can be specified in future documents, for example:

- * Send iPATCH (or PATCH) updates ([RFC8132]) to add, remove or change the links of a registration.
- * Use GET to read the currently stored set of links in a registration resource.

Those operations are out of scope of this document, and will require media types suitable for modifying sets of links.

5.3.4. Request freshness

Some security mechanisms usable with an RD allow out of order request processing, or do not even mandate replay protection at all. The RD needs to ensure that operations on the registration resource are executed in an order that does not distort the client's intentions.

This ordering of operations is expressed in terms of freshness as defined in [I-D.ietf-core-echo-request-tag]. Requests that alter a resource's state need to be fresh relative to the latest request that altered that state in a conflicting way.

An RD SHOULD determine a request's freshness, and MUST use the Echo option if it requires request freshness and can not determine the it in any other way. An endpoint MUST support the use of the Echo option. (One reason why an RD would not require freshness is when no relevant registration properties are covered by its security policies.)

5.3.4.1. Efficient use of Echo by an RD

To keep latency and traffic added by the freshness requirements to a minimum, RDs should avoid naive (sufficient but inefficient) freshness criteria.

Some simple mechanisms the RD can employ are:

- * State counter. The RD can keep a monotonous counter that increments whenever a registration changes. For every registration resource, it stores the post-increment value of that resource's last change. Requests altering them need to have at least that value encoded in their Echo option, and are otherwise rejected with a 4.01 Unauthorized and the current counter value as the Echo value. If other applications on the same server use Echo as well, that encoding may include a prefix indicating that it pertains to the RD's counter.

The value associated with a resource needs to be kept across the removal of registrations if the same registration resource is to be reused.

The counter can be reset (and the values of removed resources forgotten) when all previous security associations are reset.

This is the "Persistent Counter" method of
[I-D.ietf-core-echo-request-tag] Appendix A.

- * Preemptive Echo values. The current state counter can be sent in an Echo option not only when requests are rejected with 4.01 Unauthorized, but also with successful responses. Thus, clients can be provided with Echo values sufficient for their next request on a regular basis.

While endpoints may discard received Echo values at leisure between requests, they are encouraged to retain these values for the next request to avoid additional round trips.

- * If the RD can ensure that only one security association has modifying access to any registration at any given time, and that security association provides order on the requests, that order is sufficient to show request freshness.

5.3.4.2. Examples of Echo usage

Figure 18 shows the interactions of an endpoint that has forgotten the server's latest Echo value and temporarily reduces its registration lifetime:

Req: POST /rd/4521?lt=7200

Res: 4.01 Unauthorized
Echo: 0x0123

(EP tries again immediately)

Req: POST /rd/4521?lt=7200
Echo: 0x0123

Res: 2.04 Changed
Echo: 0x0124

(Later the EP regains its confidence in its long-term reachability)

Req: POST /rd/4521?lt=90000
Echo: 0x0124

Res: 2.04 Changed
Echo: 0x0247

Figure 18: Example update of a registration

The other examples do not show Echo options for simplicity, and because they lack the context for any example values to have meaning.

6. RD Lookup

To discover the resources registered with the RD, a lookup interface must be provided. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. JSON or CBOR link format [I-D.ietf-core-links-json]) or using more advanced interfaces (e.g. supporting context or semantic based lookup) on different resources that are discovered independently.

RD Lookup allows lookups for endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, an endpoint lookup MUST return a list of endpoints and a resource lookup MUST return a list of links to resources.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

Lookup Type	Resource Type	Mandatory
Resource	core.rd-lookup-res	Mandatory
Endpoint	core.rd-lookup-ep	Mandatory

Table 1: Lookup Types

6.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD by the registrant. The links and link parameters returned by the lookup are equal to the originally submitted ones, except that the target reference is fully resolved, and that the anchor reference is fully resolved if it is present in the lookup result at all.

Links that did not have an anchor attribute in the registration are returned without an anchor attribute. Links of which href or anchor was submitted as a (full) URI are returned with the respective attribute unmodified.

The above rules allow the client to interpret the response as links without any further knowledge of the storage conventions of the RD. The RD MAY replace the registration base URIs with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

If the base URI of a registration contains a link-local address, the RD MUST NOT show its links unless the lookup was made from the link on which the registered endpoint can be reached. The RD MUST NOT include zone identifiers in the resolved URIs.

6.2. Lookup filtering

Using the Accept Option, the requester can control whether the returned list is returned in CoRE Link Format ("application/link-format", default) or in alternate content-formats (e.g. from [I-D.ietf-core-links-json]).

Multiple search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned. The RD MUST support matching with multiple search criteria.

A link matches a search criterion if it has an attribute of the same name and the same value, allowing for a trailing "*" wildcard operator as in Section 4.1 of [RFC6690]. Attributes that are defined as "relation-types" (in the link-format ABNF) match if the search value matches any of their values (see Section 4.1 of [RFC6690]; e.g. "?if=tag:example.net,2020:sensor" matches ";if=example.regname tag:example.net,2020:sensor;"). A resource link also matches a search criterion if its endpoint would match the criterion, and vice versa, an endpoint link matches a search criterion if any of its resource links matches it.

Note that "href" is a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it. Queries for resource link targets MUST be in URI form (i.e. not relative references) and are matched against a resolved link target. Queries for endpoints SHOULD be expressed in path-absolute form if possible and MUST be expressed in URI form otherwise; the RD SHOULD recognize either. The "anchor" attribute is usable for resource lookups, and, if queried, MUST be in URI form as well.

Additional query parameters "page" and "count" are used to obtain lookup results in specified increments using pagination, where count specifies how many links to return and page specifies which subset of links organized in sequential pages, each containing 'count' links,

starting with link zero and page zero. Thus, specifying count of 10 and page of 0 will return the first 10 links in the result set (links 0-9). Count = 10 and page = 1 will return the next 'page' containing links 10-19, and so on. Unlike block-wise transfer of a complete result set, these parameters ensure that each chunk of results can be interpreted on its own. This simplifies the processing, but can result in duplicate or missed items when coinciding with changes from the registration interface.

Endpoints that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

When resource observation is used, every time the set of matching links changes, or the content of a matching link changes, the RD sends a notification with the matching link set. The notification contains the successful current response to the given request, especially with respect to representing zero matching links (see "Success" item below).

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables: type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 4.3.

search := Search criteria for limiting the number of results (optional).

The search criteria are an associative array, expressed in a form-style query as per the URI template (see [RFC6570] Sections 2.4.2 and 3.2.8)

page := Page (optional). Parameter cannot be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of

results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

Accept: absent, application/link-format or any other indicated media type representing web links

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format" or other web link payload containing matching entries for the lookup.

The payload can contain zero links (which is an empty payload in [RFC6690] link format, but could also be "[]" in JSON based formats), indicating that no entities matched the request.

6.3. Resource lookup examples

The examples in this section assume the existence of CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example resource look-up locations discovered in Figure 5:

Req: GET /rd-lookup/res?rt=tag:example.org,2020:temperature

Res: 2.05 Content

Payload:

```
<coap://[2001:db8:3::123]:61616/temp>;  
  rt="tag:example.org,2020:temperature"
```

Figure 19: Example a resource lookup

A client that wants to be notified of new resources as they show up can use observation:


```
Req: GET /rd-lookup/res?rt=tag:example.org,2020:light
Observe: 0
```

```
Res: 2.05 Content
Observe: 23
Payload: empty
```

(at a later point in time)

```
Res: 2.05 Content
Observe: 24
Payload:
<coap://[2001:db8:3::124]/west>;rt="tag:example.org,2020:light",
<coap://[2001:db8:3::124]/south>;rt="tag:example.org,2020:light",
<coap://[2001:db8:3::124]/east>;rt="tag:example.org,2020:light"
```

Figure 20: Example an observing resource lookup

The following example shows a client performing a paginated resource lookup

```
Req: GET /rd-lookup/res?page=0&count=5
```

```
Res: 2.05 Content
Payload:
<coap://[2001:db8:3::123]:61616/res/0>;ct=60,
<coap://[2001:db8:3::123]:61616/res/1>;ct=60,
<coap://[2001:db8:3::123]:61616/res/2>;ct=60,
<coap://[2001:db8:3::123]:61616/res/3>;ct=60,
<coap://[2001:db8:3::123]:61616/res/4>;ct=60
```

```
Req: GET /rd-lookup/res?page=1&count=5
```

```
Res: 2.05 Content
Payload:
<coap://[2001:db8:3::123]:61616/res/5>;ct=60,
<coap://[2001:db8:3::123]:61616/res/6>;ct=60,
<coap://[2001:db8:3::123]:61616/res/7>;ct=60,
<coap://[2001:db8:3::123]:61616/res/8>;ct=60,
<coap://[2001:db8:3::123]:61616/res/9>;ct=60
```

Figure 21: Examples of paginated resource lookup

The following example shows a client performing a lookup of all resources of all endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th response of section 5 of [RFC6690].

It demonstrates how absolute link targets stay unmodified, while relative ones are resolved:

Req: GET /rd-lookup/res?et=tag:example.com,2020:platform

Res: 2.05 Content

Payload:

```
<coap://sensor1.example.com/sensors>;ct=40;title="Sensor Index",
<coap://sensor1.example.com/sensors/temp>;rt=temperature-c;if=sensor,
<coap://sensor1.example.com/sensors/light>;rt=light-lux;if=sensor,
<http://www.example.com/sensors/t123>;rel=describedby;
  anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor1.example.com/t>;rel=alternate;
  anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor2.example.com/sensors>;ct=40;title="Sensor Index",
<coap://sensor2.example.com/sensors/temp>;rt=temperature-c;if=sensor,
<coap://sensor2.example.com/sensors/light>;rt=light-lux;if=sensor,
<http://www.example.com/sensors/t123>;rel=describedby;
  anchor="coap://sensor2.example.com/sensors/temp",
<coap://sensor2.example.com/t>;rel=alternate;
  anchor="coap://sensor2.example.com/sensors/temp"
```

Figure 22: Example of resource lookup from multiple endpoints

6.4. Endpoint lookup

The endpoint lookup returns links to and information about registration resources, which themselves can only be manipulated by the registering endpoint.

Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base; reports the registrant-ep's address if no explicit base was given) as well as a constant resource type (rt="core.rd-ep"); the lifetime (lt) is not reported. Additional endpoint attributes are added as target attributes to their endpoint link unless their specification says otherwise.

Links to endpoints SHOULD be presented in path-absolute form or, if required, as (full) URIs. (This ensures that the output conforms to Limited Link Format as described in Appendix C.)

Base addresses that contain link-local addresses MUST NOT include zone identifiers, and such registrations MUST NOT be shown unless the lookup was made from the same link from which the registration was made.

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

An RD can report registrations in lookup whose URI scheme and authority differ from the lookup resource's. Lookup clients MUST be prepared to see arbitrary URIs as registration resources in the results and treat them as opaque identifiers; the precise semantics of such links are left to future specifications.

The following example shows a client performing an endpoint lookup limited to endpoints of endpoint type
"tag:example.com,2020:platform":

Req: GET /rd-lookup/ep?et=tag:example.com,2020:platform

Res: 2.05 Content

Payload:

```
</rd/1234>;base="coap://[2001:db8:3::127]:61616";ep=node5;  
  et="tag:example.com,2020:platform";ct=40;rt=core.rd-ep,  
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep=node7;  
  et="tag:example.com,2020:platform";ct=40;d=floor-3;  
  rt=core.rd-ep
```

Figure 23: Examples of endpoint lookup

7. Security policies

The security policies that are applicable to an RD strongly depend on the application, and are not set out normatively here.

This section provides a list of aspects that applications should consider when describing their use of the RD, without claiming to cover all cases. It is using terminology of [I-D.ietf-ace-oauth-authz], in which the RD acts as the Resource Server (RS), and both registrant-eps and lookup clients act as Clients (C) with support from an Authorization Server (AS), without the intention of ruling out other (e.g. certificate / public-key infrastructure (PKI) based) schemes.

Any, all or none of the below can apply to an application. Which are relevant depends on its protection objectives.

Security policies are set by configuration of the RD, or by choice of the implementation. Lookup clients (and, where relevant, endpoints) can only trust an RD to uphold them if it is authenticated, and authorized to serve as an RD according to the application's requirements.

7.1. Endpoint name

Whenever an RD needs to provide trustworthy results to clients doing endpoint lookup, or resource lookup with filtering on the endpoint name, the RD must ensure that the registrant is authorized to use the given endpoint name. This applies both to registration and later to operations on the registration resource. It is immaterial whether the client is the registrant-ep itself or a CT is doing the registration: The RD cannot tell the difference, and CTs may use authorization credentials authorizing only operations on that particular endpoint name, or a wider range of endpoint names.

It is up to the concrete security policy to describe how endpoint name and sector are transported when certificates are used. For example, it may describe how SubjectAltName dNSName entries are mapped to endpoint and domain names.

7.1.1. Random endpoint names

Conversely, in applications where the RD does not check the endpoint name, the authorized registering endpoint can generate a random number (or string) that identifies the endpoint. The RD should then remember unique properties of the registrant, associate them with the registration for as long as its registration resource is active (which may be longer than the registration's lifetime), and require the same properties for operations on the registration resource.

Registrants that are prepared to pick a different identifier when their initial attempt (or attempts, in the unlikely case of two subsequent collisions) at registration is unauthorized should pick an identifier at least twice as long as the expected number of registrants; registrants without such a recovery options should pick significantly longer endpoint names (e.g. using UUID URNs [RFC4122]).

7.2. Entered resources

When lookup clients expect that certain types of links can only originate from certain endpoints, then the RD needs to apply filtering to the links an endpoint may register.

For example, if clients use an RD to find a server that provides firmware updates, then any registrant that wants to register (or update) links to firmware sources will need to provide suitable credentials to do so, independently of its endpoint name.

Note that the impact of having undesirable links in the RD depends on the application: if the client requires the firmware server to present credentials as a firmware server, a fraudulent link's impact is limited to the client revealing its intention to obtain updates and slowing down the client until it finds a legitimate firmware server; if the client accepts any credentials from the server as long as they fit the provided URI, the impact is larger.

An RD may also require that links are only registered if the registrant is authorized to publish information about the anchor (or even target) of the link. One way to do this is to demand that the registrant present the same credentials as a client that they'd need to present if contacted as a server at the resources' URI, which may include using the address and port that are part of the URI. Such a restriction places severe practical limitations on the links that can be registered.

As above, the impact of undesirable links depends on the extent to which the lookup client relies on the RD. To avoid the limitations, RD applications should consider prescribing that lookup clients only use the discovered information as hints, and describe which pieces of information need to be verified because they impact the application's security. A straightforward way to verify such information is to request it again from an authorized server, typically the one that hosts the target resource. That similar to what happens in Section 4.3 when the URI discovery step is repeated.

7.3. Link confidentiality

When registrants publish information in the RD that is not available to any client that would query the registrant's /.well-known/core interface, or when lookups to that interface are subject to stricter firewalling than lookups to the RD, the RD may need to limit which lookup clients may access the information.

In this case, the endpoint (and not the lookup clients) needs to be careful to check the RD's authorization. The RD needs to check any lookup client's authorization before revealing information directly (in resource lookup) or indirectly (when using it to satisfy a resource lookup search criterion).

7.4. Segmentation

Within a single RD, different security policies can apply.

One example of this are multi-tenant deployments separated by the sector (d) parameter. Some sectors might apply limitations on the endpoint names available, while others use a random identifier approach to endpoint names and place limits on the entered links based on their attributes instead.

Care must be taken in such setups to determine the applicable access control measures to each operation. One easy way to do that is to mandate the use of the sector parameter on all operations, as no credentials are suitable for operations across sector borders anyway.

7.5. First-Come-First-Remembered: A default policy

The First-Come-First-Remembered policy is provided both as a reference example for a security policy definition, and as a policy that implementations may choose to use as default policy in absence of other configuration. It is designed to enable efficient discovery operations even in ad-hoc settings.

Under this policy, the RD accepts registrations for any endpoint name that is not assigned to an active registration resource, and only accepts registration updates from the same endpoint. The policy is minimal in that towards lookup clients it does not make any of the claims of Section 7.2 and Section 7.3, and its claims on Section 7.1 are limited to the lifetime of that endpoint's registration. It does, however, guarantee towards any endpoint that for the duration of its registration, its links will be discoverable on the RD.

When a registration or operation is attempted, the RD MUST determine the client's subject name or public key:

- * If the client's credentials indicate any subject name that is certified by any authority which the RD recognizes (which may be the system's trust anchor store), all such subject names are stored. With CWT or JWT based credentials (as common with ACE), the Subject (sub) claim is stored as a single name, if it exists. With X.509 certificates, the Common Name (CN) and the complete list of SubjectAltName entries are stored. In both cases, the authority that certified the claim is stored along with the subject, as the latter may only be locally unique.

- * Otherwise, if the client proves possession of a private key, the matching public key is stored. This applies both to raw public keys and to the public keys indicated in certificates that failed the above authority check.
- * If neither is present, a reference to the security session itself is stored. With (D)TLS, that is the connection itself, or the session resumption information if available. With OSCORE, that is the security context.

As part of the registration operation, that information is stored along with the registration resource.

The RD MUST accept all registrations whose registration resource is not already active, as long as they are made using a security layer supported by the RD.

Any operation on a registration resource, including registrations that lead to an existing registration resource, MUST be rejected by the RD unless all the stored information is found in the new request's credentials.

Note that even though subject names are compared in this policy, they are never directly compared to endpoint names, and an endpoint can not expect to "own" any particular endpoint name outside of an active registration -- even if a certificate says so. It is an accepted shortcoming of this approach that the endpoint has no indication of whether the RD remembers it by its subject name or public key; recognition by subject happens on a best-effort base (given the RD may not recognize any authority). Clients MUST be prepared to pick a different endpoint name when rejected by the RD initially or after a change in their credentials; picking an endpoint name as per Section 7.1.1 is an easy option for that.

For this policy to be usable without configuration, clients should not set a sector name in their registrations. An RD can set a default sector name for registrations accepted under this policy, which is useful especially in a segmented setup where different policies apply to different sectors. The configuration of such a behavior, as well as any other configuration applicable to such an RD (i.e. the set of recognized authorities) is out of scope for this document.

8. Security Considerations

The security considerations as described in Section 5 of [RFC8288] and Section 6 of [RFC6690] apply. The `"/.well-known/core"` resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252].

Access that is limited or affects sensitive data SHOULD be protected, e.g. using (D)TLS or OSCORE ([RFC8613]; which aspects of the RD this affects depends on the security policies of the application (see Section 7).

8.1. Discovery

Most steps in discovery of the RD, and possibly its resources, are not covered by CoAP's security mechanisms. This will not endanger the security properties of the registrations and lookup itself (where the client requires authorization of the RD if it expects any security properties of the operation), but may leak the client's intention to third parties, and allow them to slow down the process.

To mitigate that, clients can retain the RD's address, use secure discovery options like configured addresses, and send queries for RDs in a very general form (`"?rt=core.rd"` rather than `"?rt=core.rd-lookup-ep"`).

8.2. Endpoint Identification and Authentication

An Endpoint (name, sector) pair is unique within the set of endpoints registered by the RD. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint on an RD SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security.

Consider the following threat: two devices A and B are registered at a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the DTLS exchange. Then, it specifies the endpoint name of device B as the name of its own endpoint in device A. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Endpoint authorization needs to be checked on registration and registration resource operations independently of whether there are configured requirements on the credentials for a given endpoint name (and sector; Section 7.1) or whether arbitrary names are accepted (Section 7.1.1).

Simple registration could be used to circumvent address-based access control: An attacker would send a simple registration request with the victim's address as source address, and later look up the victim's /.well-known/core content in the RD. Mitigation for this is recommended in Section 5.1.

The registration resource path is visible to any client that is allowed endpoint lookup, and can be extracted by resource lookup clients as well. The same goes for registration attributes that are shown as target attributes or lookup attributes. The RD needs to consider this in the choice of registration resource paths, and administrators or endpoint in their choice of attributes.

8.3. Access Control

Access control SHOULD be performed separately for the RD registration and Lookup API paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the sector, endpoint or resource level.

The precise access controls necessary (and the consequences of failure to enforce them) depend on the protection objectives of the application and the security policies (Section 7) derived from them.

8.4. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly amplify and distribute a DoS attack as UDP does not require return routability check. Since RD lookup responses can be significantly larger than requests, RDs are prone to this.

[RFC7252] describes this at length in its Section 11.3, including some mitigation by using small block sizes in responses. The upcoming [I-D.ietf-core-echo-request-tag] updates that by describing a source address verification mechanism using the Echo option.

[If this document is published together with or after I-D.ietf-core-echo-request-tag, the above paragraph is replaced with the following:

[RFC7252] describes this at length in its Section 11.3, and [I-D.ietf-core-echo-request-tag] (which updates the former) recommends using the Echo option to verify the request's source address.

]

8.5. Skipping freshness checks

When RD based applications are built in which request freshness checks are not performed, these concerns need to be balanced:

- * When alterations to registration attributes are reordered, an attacker may create any combination of attributes ever set, with the attack difficulty determined by the security layer's replay properties.

For example, if Figure 18 were conducted without freshness assurances, an attacker could later reset the lifetime back to 7200. Thus, the device is made unreachable to lookup clients.

- * When registration updates without query parameters (which just serve to restart the lifetime) can be reordered, an attacker can use intercepted messages to give the appearance of the device being alive to the RD.

This is unacceptable when the RD's security policy promises reachability of endpoints (e.g. when disappearing devices would trigger further investigation), but may be acceptable with other policies.

9. IANA Considerations

9.1. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values sub-registry of the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690]:

Value	Description	Reference
core.rd	Directory resource of an RD	RFCTHIS Section 4.3
core.rd-lookup-res	Resource lookup of an RD	RFCTHIS Section 4.3
core.rd-lookup-ep	Endpoint lookup of an RD	RFCTHIS Section 4.3
core.rd-ep	Endpoint resource of an RD	RFCTHIS Section 6

Table 2

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the sub-registry "IPv6 Neighbor Discovery Option Formats" of the "Internet Control Message Protocol version 6 (ICMPv6) Parameters" registry:

- * Resource Directory Address Option (TBD38)

[The RFC editor is asked to replace TBD38 with the assigned number in the document; the value 38 is suggested.]

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include

- * the human readable name of the parameter,
- * the short name as used in query parameters or target attributes,
- * indication of whether it can be passed as a query parameter at registration of endpoints, as a query parameter in lookups, or be expressed as a target attribute,
- * syntax and validity requirements if any,

- * a description,
- * and a link to reference documentation.

The query parameter MUST be both a valid URI query key [RFC3986] and a token as used in [RFC8288].

The description must give details on whether the parameter can be updated, and how it is to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

Full name	Short	Validity	Use	Description
Endpoint Name	ep	Unicode*	RLA	Name of the endpoint
Lifetime	lt	1-4294967295	R	Lifetime of the registration in seconds
Sector	d	Unicode*	RLA	Sector to which this endpoint belongs
Registration Base URI	base	URI	RLA	The scheme, address and port and path at which this server is available
Page	page	Integer	L	Used for pagination
Count	count	Integer	L	Used for pagination
Endpoint Type	et	Section 9.3.1	RLA	Semantic type of the endpoint (see Section 9.4)

Table 3: RD Parameters

(Short: Short name used in query parameters or target attributes.
 Validity: Unicode* = 63 Bytes of UTF-8 encoded Unicode, with no control characters as per Section 5. Use: R = used at registration, L = used at lookup, A = expressed in target attribute.)

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document. All their reference documentation entries point to this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (E.g. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used target attributes (For

example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] target attribute).

9.3.1. Full description of the "Endpoint Type" RD Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type sub-registry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, RDs implementing this specification automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called '"Endpoint Type" (et=) RD Parameter values'. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

- * The values MUST be related to the purpose described in Section 9.3.1.
- * The registered values MUST conform to the ABNF reg-rel-type definition of [RFC6690] and MUST NOT be a URI.
- * It is recommended to use the period "." character for segmentation.

The registry initially contains one value:

- * "core.rd-group": An application group as described in Appendix A.

9.5. Multicast Address Registration

IANA is asked to assign the following multicast addresses for use by CoAP nodes:

IPv4 -- "all CoRE Resource Directories" address MCD2 (suggestion: 224.0.1.189), from the "IPv4 Multicast Address Space Registry". As the address is used for discovery that may span beyond a single network, it has come from the Internetwork Control Block (224.0.1.x) [RFC5771].

IPv6 -- "all CoRE Resource Directories" address MCD1 (suggestions FF0X::FE), from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.

[The RFC editor is asked to replace MCD1 and MCD2 with the assigned addresses throughout the document.]

9.6. Well-Known URIs

IANA is asked to permanently register the URI suffix "rd" in the "Well-Known URIs" registry. The change controller is the IETF, this document is the reference.

9.7. Service Names and Transport Protocol Port Number Registry

IANA is asked to enter four new items into the Service Names and Transport Protocol Port Number Registry:

- * Service name: "core-rd", Protocol: "udp", Description: "Resource Directory accessed using CoAP"
- * Service name "core-rd-dtls", Protocol: "udp", Description: "Resource Directory accessed using CoAP over DTLS"
- * Service name: "core-rd", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TCP"
- * Service name "core-rd-tls", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TLS"

All in common have this document as their reference.

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LwM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the RD with a CoAP interface to facilitate the installation and start-up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address as described in Appendix A. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative.

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infrastructure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and the sensor. The addresses shown in Table 4 below stand in for these in the following examples.

Name	IPv6 address
luminary1	2001:db8:4::1
luminary2	2001:db8:4::2
Presence sensor	2001:db8:4::3
RD	2001:db8:4::ff

Table 4: Addresses used in the examples

In Section 10.1.2 the use of RD during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have `rt=tag:example.com,2020:light`, and the presence sensor has `rt=tag:example.com,2020:p-sensor`. The endpoints have names which are relevant to the light installation manager. In this case `luminary1`, `luminary2`, and the presence sensor are located in room 2-4-015, where `luminary1` is located at the window and `luminary2` and the presence sensor are located at the door. The endpoint names reflect this physical location. The middle, left and right lamps are accessed via path `/light/middle`, `/light/left`, and `/light/right` respectively. The identifiers relevant to the RD are shown in Table 5 below:

Name	endpoint	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/middle	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/right	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/left	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/middle	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/right	tag:example.com,2020:light
Presence sensor	ps_R2-4-015_door	/ps	tag:example.com,2020:p-sensor

Table 5: RD identifiers

It is assumed that the CT has performed RD discovery and has received a response like the one in the Section 4.3 example.

The CT inserts the endpoints of the luminaries and the sensor in the RD using the registration base URI parameter (base) to specify the interface address:

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=lm_R2-4-015_wndw&base=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/4521

Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=lm_R2-4-015_door&base=coap://[2001:db8:4::2]&d=R2-4-015
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/4522

Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=ps_R2-4-015_door&base=coap://[2001:db8:4::3]&d=R2-4-015
Payload:
</ps>;rt="tag:example.com,2020:p-sensor"

Res: 2.01 Created
Location-Path: /rd/4523
```

Figure 24: Example of registrations a CT enters into an RD

The sector name d=R2-4-015 has been added for an efficient lookup because filtering on "ep" name is more awkward. The same sector name is communicated to the two luminaries and the presence sensor by the CT.

The group is specified in the RD. The base parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, the resources supported by all group members are published.

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::1]
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/501
```

Figure 25: Example of a multicast group a CT enters into an RD

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its sector and being configured to join any group containing lights, searches for candidate groups and joins them:

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
      ?d=R2-4-015&et=core.rd-group&rt=light

Res: 2.05 Content
Payload:
</rd/501>;ep=grp_R2-4-015;et=core.rd-group;
      base="coap://[ff05::1]";rt=core.rd-ep
```

Figure 26: Example of a lookup exchange to find suitable multicast addresses

From the returned base parameter value, the luminary learns the multicast address of the multicast group.

The presence sensor can learn the presence of groups that support resources with `rt=tag:example.com,2020:light` in its own sector by sending the same request, as used by the luminary. The presence sensor learns the multicast address to use for sending messages to the luminaries.

10.2. OMA Lightweight M2M (LwM2M)

OMA LwM2M is a profile for device services based on CoAP, providing interfaces and operations for device management and device service enablement.

An LwM2M server is an instance of an LwM2M middleware service layer, containing an RD ([LwM2M] page 36f).

That RD only implements the registration interface, and no lookup is implemented. Instead, the LwM2M server provides access to the registered resources, in a similar way to a reverse proxy.

The location of the LwM2M Server and RD URI path is provided by the LwM2M Bootstrap process, so no dynamic discovery of the RD is used. LwM2M Servers and endpoints are not required to implement the /.well-known/core resource.

11. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, Jan Newmarch, Matthias Kovatsch, Jaime Jimenez and Ted Lemon have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the RD concepts were originally developed.

12. Changelog

changes from -27 to -28

- * Security policies / link confidentiality: Point out the RD's obligations that follow from such a policy.
- * Simple registration: clarify term "regular registration" by introducing it along with the reference to Section 5
- * Wording fix in first-come-first-remembered
- * Wording fixes in RD definition
- * Capitalization: Consistently using "registration resource"

changes from -26 to -27

- * In general, this addresses the points that were pointed out in <https://mailarchive.ietf.org/arch/msg/core/xWLomwwhovkU-CPGNxnvs40BhaM/> as having "evolved from the review comments being discussed in the interim meetings", and the review comments from Esko Dijk that were largely entangled in these points.
- * Relaxation of the serialization rules for link-format

The interpretation of RFC6690 used in Appendix B.4 was shown to be faulty. Along with a correction, the common implementations of link-format were surveyed again and it was found that the only one

that employed the faulty interpretation can still safely be upgraded. These were removed from the set considered for Limited Link Format, making the set of valid Limited Link Format documents larger.

As a consequence, the prescribed serialization of RD output can be roughly halved in bytes.

There might be additional usage patterns that are possible with the new set of constraints, but there is insufficient implementation and deployment experience with them to warrant a change changes on that front at this point. The specification can later be extended compatibly to allow these cases and drop the requirement of Limited Link Format.

- * Add Request freshness subsection

It is now recommended (with security considerations on consequences of not doing it) to require ordering of RD operations.

The Echo mechanism (previously suggested in various places but never exclusively) is the one prescribed way of getting this ordering, making the echo-request-tag reference normative.

- * Improved expression about when an RD needs to verify simple registration.

The simple wording missed the authorization part, and did not emphasize that this is a per-deployment property.

- * Point out the non-atomic properties of paginated access.

- * Clarification around impl-info reference.

- * Inconsistencies and extraneous quotations removed from examples.

changes from -25 to -26

- * Security policies:

- The First-Come-First-Remembered policy is added as an example and a potential default behavior.
- Clarify that the mapping between endpoint names and subject fields is up to a policy that defines reliance on names, and give an example.

- Random EP names: Point that multiple collisions are possible but unlikely.
- Add pointers to policies:
 - o RD replication: Point out that policies may limit that.
 - o Registration: Reword (ep, d) mapping to a previous registration's resource that could have been read as another endpoint taking over an existing registration.
- Clarify that the security policy is a property of the RD the any client may need to verify by checking the RD's authorization.
- Clarify how information from an untrusted RD can be verified
- Remove speculation about how in detail ACE scopes are obtained.
- * Security considerations:
 - Generalize to all current options for security layers usable with CoAP (OSCORE was missing as the text predated RFC8613)
 - Relax the previous SHOULD on secure access to SHOULD where protection is indicated by security policies (bringing the text in line with the -25 changes)
 - Point out that failure to follow the security considerations has implications depending on the protection objective described with the security policies
 - Shorten amplification mitigation
 - Add note about information in Registration Resource path.
 - Acknowledge that most host discovery operations are not secured; mention consequences and mitigation.
- * Abstract, introduction: removed "or disperse networks"
- * RD discovery:
 - Drop the previously stated assumption that RDAO and any DHCP options would only be used together with SLAAC and DHCP for address configuration, respectively.

- Give concrete guidance for address selection based on RFC6724 when responding to multicasts
- RDAO:
 - o Clarify that it is an option for RAs and not other ND messages.
 - o Change Lifetime from 16-bit minutes to 32-bit seconds and swap it with Reserved (aligning it with RDNSS which it shares other properties as well).
- Point out that clients may need to check RD authorization already in last discovery step
- * Registration:
 - Wording around "mostly mandatory" has been improved, conflicts clarified and sector default selection adjusted.
- * Simple registration: Rather than coopting POSTs to /.well-known/core, a new resource /.well-known/rd is registered. A historical note in the text documents the change.
- * Examples:
 - Use example URIs rather than unclear reg names (unless it's RFC6690 examples, which were kept for continuity)
 - The LwM2M example was reduced from an outdated explanation of the complete LwM2M model to a summary of how RD is used in there, with a reference to the current specification.
 - Luminary example: Explain example addresses
 - Luminary example: Drop reference to coap-group mechanism that's becoming obsolete, and thus also to RFC7390
 - Multicast addresses in the examples were changed from ff35:30:2001:db8::x to ff35:30:2001:db8:f1::8000:x; the 8000 is to follow RFC 3307, and the f1 is for consistency with all the other example addresses where 2001:db8::/32 is subnetted to 2001:db8:x::/48 by groups of internally consistent examples.
- * Use case text enhancements
 - Home and building automation: Tie in with RD

- M2M: Move system design paragraph towards the topic of reusability.
- * Various editorial fixes in response to Gen-ART and IESG reviews.
- * Rename 'Full description of the "Endpoint Type" Registration Parameter' section to '... RD Parameter'
- * Error handling: Place a SHOULD around the likely cases, and make the previous "MUST to the best of their capabilities" a "must".
- * impl-info: Add note about the type being WIP
- * Interaction tables: list CTs as possible initiators where applicable
- * Registration update: Relax requirement to not send parameters needlessly
- * Terminology: Clarify that the CTs' installation events can occur multiple times.
- * Promote RFCs 7252, 7230 and 8288 to normative references
- * Moved Christian Amsuess to first author

changes from -24 to -25

- * Large rework of section 7 (Security policies)

Rather than prescribing which data in the RD is authenticated (and how), it now describes what applications built on an RD can choose to authenticate, show possibilities on how to do it and outline what it means for clients.

This addresses Russ' Genart review points on details in the text in a rather broad fashion. That is because the discussion on the topic inside the WG showed that that text on security has been driven more review-by-review than by an architectural plan of the authors and WG.

- * Add concrete suggestions (twice as long as registrant number with retries, or UUIDs without) for random endpoint names
- * Point out that simple registration can have faked origins, RECOMMEND mitigation when applicable and suggest the Echo mechanism to implement it.

- * Reference existing and upcoming specifications for DDOS mitigation in CoAP.
 - * Explain the provenance of the example's multicast address.
 - * Make "SHOULD" of not manipulating foreign registrations a "should" and explain how it is enforced
 - * Clarify application of RFC6570 to search parameters
 - * Syntactic fixes in examples
 - * IANA:
 - Don't announce expected number of registrations (goes to write-up)
 - Include syntax as part of a field's validity in entry requirements
 - * Editorial changes
 - Align wording between abstract and introduction
 - Abbreviation normalization: "ER model", "RD"
 - RFC8174 boilerplate update
 - Minor clarity fixes
 - Markup and layouting
- changes from -23 to -24
- * Discovery using DNS-SD added again
 - * Minimum lifetime (lt) reduced from 60 to 1
 - * References added
 - * IANA considerations
 - added about .well-known/core resource
 - added DNS-SD service names
 - made RDAO option number a suggestion

- added "reference" field to endpoint type registry
 - * Lookup: mention that anchor is a legitimate lookup attribute
 - * Terminology and example fixes
 - * Layout fixes, esp. the use of non-ASCII characters in figures
- changes from -22 to -23
- * Explain that updates can not remove attributes
 - * Typo fixes
- changes from -21 to -22
- * Request a dedicated IPv4 address from IANA (rather than sharing with All CoAP nodes)
 - * Fix erroneous examples
 - * Editorial changes
- Add figure numbers to examples
 - Update RD parameters table to reflect changes of earlier versions in the text
 - Typos and minor wording
- changes from -20 to -21
- (Processing comments during WGLC)
- * Defer outdated description of using DNS-SD to find an RD to the defining document
 - * Describe operational conditions in automation example
 - * Recommend particular discovery mechanisms for some managed network scenarios
- changes from -19 to -20
- (Processing comments from the WG chair review)
- * Define the permissible characters in endpoint and sector names

- * Express requirements on NAT situations in more abstract terms
- * Shifted heading levels to have the interfaces on the same level
- * Group instructions for error handling into general section
- * Simple Registration: process reflowed into items list
- * Updated introduction to reflect state of CoRE in general, reference RFC7228 (defining "constrained") and use "IoT" term in addition to "M2M"
- * Update acknowledgements
- * Assorted editorial changes
 - Unify examples style
 - Terminology: RDAO defined and not only expanded
 - Add CT to Figure 1
 - Consistency in the use of the term "Content Format"

changes from -18 to -19

- * link-local addresses: allow but prescribe split-horizon fashion when used, disallow zone identifiers
- * Remove informative references to documents not mentioned any more

changes from -17 to -18

- * Rather than re-specifying link format (Modernized Link Format), describe a Limited Link Format that's the uncontested subset of Link Format
- * Acknowledging the -17 version as part of the draft
- * Move "Read endpoint links" operation to future specification like PATCH
- * Demote links-json to an informative reference, and removed them from exchange examples
- * Add note on unusability of link-local IP addresses, and describe mitigation.

- * Reshuffling of sections: Move additional operations and endpoint lookup back from appendix, and groups into one
- * Lookup interface tightened to not imply applicability for non link-format lookups (as those can have vastly different views on link cardinality)
- * Simple registration: Change sequence of GET and POST-response, ensuring unsuccessful registrations are reported as such, and suggest how devices that would have required the inverse behavior can still cope with it.
- * Abstract and introduction reworded to avoid the impression that resources are stored in full in the RD
- * Simplify the rules governing when a registration resource can or must be changed.
- * Drop a figure that has become useless due to the changes of and -13 and -17
- * Wording consistency fixes: Use "Registrations" and "target attributes"
- * Fix incorrect use of content negotiation in discovery interface description (Content-Format -> Accept)
- * State that the base attribute value is part of endpoint lookup even when implicit in the registration
- * Update references from RFC5988 to its update RFC8288
- * Remove appendix on protocol-negotiation (which had a note to be removed before publication)

changes from -16 to -17

(Note that -17 is published as a direct follow-up to -16, containing a single change to be discussed at IETF103)

- * Removed groups that are enumerations of registrations and have dedicated mechanism
- * Add groups that are enumerations of shared resources and are a special case of endpoint registrations

changes from -15 to -16

- * Recommend a common set of resources for members of a group
- * Clarified use of multicast group in lighting example
- * Add note on concurrent registrations from one EP being possible but not expected
- * Refresh web examples appendix to reflect current use of Modernized Link Format
- * Add examples of URIs where Modernized Link Format matters
- * Editorial changes

changes from -14 to -15

- * Rewrite of section "Security policies"
- * Clarify that the "base" parameter text applies both to relative references both in anchor and href
- * Renamed "Registree-EP" to Registrant-EP"
- * Talk of "relative references" and "URIs" rather than "relative" and "absolute" URIs. (The concept of "absolute URIs" of [RFC3986] is not needed in RD).
- * Fixed examples
- * Editorial changes

changes from -13 to -14

- * Rename "registration context" to "registration base URI" (and "con" to "base") and "domain" to "sector" (where the abbreviation "d" stays for compatibility reasons)
- * Introduced resource types core.rd-ep and core.rd-gp
- * Registration management moved to appendix A, including endpoint and group lookup
- * Minor editorial changes
 - PATCH/iPATCH is clearly deferred to another document
 - Recommend against query / fragment identifier in con=

- Interface description lists are described as illustrative
- Rewording of Simple Registration
- * Simple registration carries no error information and succeeds immediately (previously, sequence was unspecified)
- * Lookup: href are matched against resolved values (previously, this was unspecified)
- * Lookup: lt are not exposed any more
- * con/base: Paths are allowed
- * Registration resource locations can not have query or fragment parts
- * Default life time extended to 25 hours
- * clarified registration update rules
- * lt-value semantics for lookup clarified.
- * added template for simple registration

changes from -12 to -13

- * Added "all resource directory" nodes MC address
- * Clarified observation behavior
- * version identification
- * example rt= and et= values
- * domain from figure 2
- * more explanatory text
- * endpoints of a groups hosted by different RD
- * resolve RFC6690-vs-8288 resolution ambiguities:
 - require registered links not to be relative when using anchor
 - return absolute URIs in resource lookup

changes from -11 to -12

- * added Content Model section, including ER diagram
- * removed domain lookup interface; domains are now plain attributes of groups and endpoints
- * updated chapter "Finding a Resource Directory"; now distinguishes configuration-provided, network-provided and heuristic sources
- * improved text on: atomicity, idempotency, lookup with multiple parameters, endpoint removal, simple registration
- * updated LWM2M description
- * clarified where relative references are resolved, and how context and anchor interact
- * new appendix on the interaction with RFCs 6690, 5988 and 3986
- * lookup interface: group and endpoint lookup return group and registration resources as link targets
- * lookup interface: search parameters work the same across all entities
- * removed all methods that modify links in an existing registration (POST with payload, PATCH and iPATCH)
- * removed plurality definition (was only needed for link modification)
- * enhanced IANA registry text
- * state that lookup resources can be observable
- * More examples and improved text

changes from -09 to -10

- * removed "ins" and "exp" link-format extensions.
- * removed all text concerning DNS-SD.
- * removed inconsistency in RDAO text.
- * suggestions taken over from various sources
- * replaced "Function Set" with "REST API", "base URI", "base path"

- * moved simple registration to registration section

changes from -08 to -09

- * clarified the "example use" of the base RD resource values /rd, /rd-lookup, and /rd-group.
- * changed "ins" ABNF notation.
- * various editorial improvements, including in examples
- * clarifications for RDAO

changes from -07 to -08

- * removed link target value returned from domain and group lookup types
- * Maximum length of domain parameter 63 bytes for consistency with group
- * removed option for simple POST of link data, don't require a .well-known/core resource to accept POST data and handle it in a special way; we already have /rd for that
- * add IPv6 ND Option for discovery of an RD
- * clarify group configuration section 6.1 that endpoints must be registered before including them in a group
- * removed all superfluous client-server diagrams
- * simplified lighting example
- * introduced Commissioning Tool
- * RD-Look-up text is extended.

changes from -06 to -07

- * added text in the discovery section to allow content format hints to be exposed in the discovery link attributes
- * editorial updates to section 9
- * update author information
- * minor text corrections

Changes from -05 to -06

- * added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

- * added Update Endpoint Links using PATCH
- * http access made explicit in interface specification
- * Added http examples

Changes from -03 to -04:

- * Added http response codes
- * Clarified endpoint name usage
- * Add application/link-format+cbor content-format

Changes from -02 to -03:

- * Added an example for lighting and DNS integration
- * Added an example for RD use in OMA LWM2M
- * Added Read Links operation for link inspection by endpoints
- * Expanded DNS-SD section
- * Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

- * Added a catalogue use case.
- * Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.
- * Additional examples section added for more complex use cases.
- * New DNS-SD mapping section.
- * Added text on endpoint identification and authentication.
- * Error code 4.04 added to Registration Update and Delete requests.

- * Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- * Removed the ETag validation feature.
- * Place holder for the DNS-SD mapping section.
- * Explicitly disabled GET or POST on returned Location.
- * New registry for RD parameters.
- * Added support for the JSON Link Format.
- * Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- * Updated the version and date.

Changes from -04 to -05:

- * Restricted Update to parameter updates.
- * Added pagination support for the Lookup interface.
- * Minor editing, bug fixes and reference updates.
- * Added group support.
- * Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- * Added the ins= parameter back for the DNS-SD mapping.
- * Integrated the Simple Directory Discovery from Carsten.
- * Editorial improvements.
- * Fixed the use of ETags.
- * Fixed tickets 383 and 372

Changes from -02 to -03:

- * Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- * Updated REST interface descriptions to use RFC6570 URI Template format.
- * Introduced an improved RD Lookup design as its own function set.
- * Improved the security considerations section.
- * Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

- * Added a terminology section.
- * Changed the inclusion of an ETag in registration or update to a MAY.
- * Added the concept of an RD Domain and a registration parameter for it.
- * Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- * Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

13. References

13.1. Normative References

- [I-D.ietf-core-echo-request-tag]
Amsüss, C., Mattsson, J. P., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", Work in Progress, Internet-Draft, draft-ietf-core-echo-request-tag-12, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-echo-request-tag-12.txt>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

13.2. Informative References

- [ER] Chen, P., "The entity-relationship model--toward a unified view of data", DOI 10.1145/320434.320440, ACM Transactions on Database Systems Vol. 1, pp. 9-36, March 1976, <<https://doi.org/10.1145/320434.320440>>.

[I-D.bormann-t2trg-rel-impl]

Bormann, C., "impl-info: A link relation type for disclosing implementation information", Work in Progress, Internet-Draft, draft-bormann-t2trg-rel-impl-02, 27 September 2020, <<https://www.ietf.org/archive/id/draft-bormann-t2trg-rel-impl-02.txt>>.

[I-D.hartke-t2trg-coral]

Hartke, K., "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-hartke-t2trg-coral-09, 8 July 2019, <<https://www.ietf.org/archive/id/draft-hartke-t2trg-coral-09.txt>>.

[I-D.ietf-ace-oauth-authz]

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-37, 4 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-37.txt>>.

[I-D.ietf-core-links-json]

LI, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", Work in Progress, Internet-Draft, draft-ietf-core-links-json-10, 26 February 2018, <<https://www.ietf.org/archive/id/draft-ietf-core-links-json-10.txt>>.

[I-D.ietf-core-rd-dns-sd]

Stok, P. V. D., Koster, M., and C. Amsüss, "CoRE Resource Directory: DNS-SD mapping", Work in Progress, Internet-Draft, draft-ietf-core-rd-dns-sd-05, 7 July 2019, <<https://www.ietf.org/archive/id/draft-ietf-core-rd-dns-sd-05.txt>>.

[I-D.silverajan-core-coap-protocol-negotiation]

Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", Work in Progress, Internet-Draft, draft-silverajan-core-coap-protocol-negotiation-09, 2 July 2018, <<https://www.ietf.org/archive/id/draft-silverajan-core-coap-protocol-negotiation-09.txt>>.

[LwM2M]

Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification: Transport Bindings (Candidate Version 1.1)", 12 June 2018,

- <https://openmobilealliance.org/RELEASE/LightweightM2M/V1_1-20180612-C/OMA-TS-LightweightM2M_Transport-V1_1-20180612-C.pdf>.
- [RFC3306] Haberman, B. and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses", RFC 3306, DOI 10.17487/RFC3306, August 2002, <<https://www.rfc-editor.org/info/rfc3306>>.
- [RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, DOI 10.17487/RFC3849, July 2004, <<https://www.rfc-editor.org/info/rfc3849>>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/info/rfc4122>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC5771] Cotton, M., Vegoda, L., and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments", BCP 51, RFC 5771, DOI 10.17487/RFC5771, March 2010, <<https://www.rfc-editor.org/info/rfc5771>>.
- [RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <<https://www.rfc-editor.org/info/rfc6724>>.
- [RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <<https://www.rfc-editor.org/info/rfc6775>>.
- [RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <<https://www.rfc-editor.org/info/rfc6874>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.

- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <<https://www.rfc-editor.org/info/rfc8106>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <<https://www.rfc-editor.org/info/rfc8141>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

Appendix A. Groups Registration and Lookup

The RD-Groups usage pattern allows announcing application groups inside an RD.

Groups are represented by endpoint registrations. Their base address is a multicast address, and they SHOULD be entered with the endpoint type "core.rd-group". The endpoint name can also be referred to as a group name in this context.

The registration is inserted into the RD by a Commissioning Tool, which might also be known as a group manager here. It performs third party registration and registration updates.

The links it registers SHOULD be available on all members that join the group. Depending on the application, members that lack some resource MAY be permissible if requests to them fail gracefully.

The following example shows a CT registering a group with the name "lights" which provides two resources. The directory resource path /rd is an example RD location discovered in a request similar to Figure 5. The group address in the example is constructed from [RFC3849]'s reserved 2001:db8:: prefix as a unicast-prefix based site-local address (see [RFC3306]).


```
Req: POST coap://rd.example.com/rd?ep=lights&et=core.rd-group
      &base=coap://[ff35:30:2001:db8:f1::8000:1]
Content-Format: 40
Payload:
</light>;rt="tag:example.com,2020:light";
      if="tag:example.net,2020:actuator",
</color-temperature>;if="tag:example.net,2020:parameter";u=K

Res: 2.01 Created
Location-Path: /rd/12
```

Figure 27: Example registration of a group

In this example, the group manager can easily permit devices that have no writable color-temperature to join, as they would still respond to brightness changing commands. Had the group instead contained a single resource that sets brightness and color temperature atomically, endpoints would need to support both properties.

The resources of a group can be looked up like any other resource, and the group registrations (along with any additional registration parameters) can be looked up using the endpoint lookup interface.

The following example shows a client performing an endpoint lookup for all groups.

```
Req: GET /rd-lookup/ep?et=core.rd-group

Res: 2.05 Content
Payload:
</rd/12>;ep=lights&et=core.rd-group;
      base="coap://[ff35:30:2001:f1:db8::8000:1]";rt=core.rd-ep
```

Figure 28: Example lookup of groups

The following example shows a client performing a lookup of all resources of all endpoints (groups) with et=core.rd-group.

```
Req: GET /rd-lookup/res?et=core.rd-group

Res: 2.05 Content
Payload:
<coap://[ff35:30:2001:db8:f1::8000:1]/light>;
      rt="tag:example.com,2020:light";
      if="tag:example.net,2020:actuator",
<coap://[ff35:30:2001:db8:f1::8000:1]/color-temperature>;
      if="tag:example.net,2020:parameter";u=K,
```

Figure 29: Example lookup of resources inside groups

Appendix B. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC8288] which defines Link header fields, and [RFC7252] providing the transport). This appendix summarizes the mechanisms and semantics at play from an entry in `"/.well-known/core"` to a resource lookup.

This text is primarily aimed at people entering the field of Constrained Restful Environments from applications that previously did not use web mechanisms.

B.1. A simple example

Let's start this example with a very simple host, `"2001:db8:f0::1"`. A client that follows classical CoAP Discovery ([RFC7252] Section 7), sends the following multicast request to learn about neighbours supporting resources with resource-type `"temperature"`.

The client sends a link-local multicast:

```
Req: GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature
```

```
Res: 2.05 Content
```

```
Payload:
```

```
</sensors/temp>;rt=temperature;ct=0
```

Figure 30: Example of direct resource discovery

where the response is sent by the server, `"[2001:db8:f0::1]:5683"`.

While the client -- on the practical or implementation side -- can just go ahead and create a new request to `"[2001:db8:f0::1]:5683"` with Uri-Path: `"sensors"` and `"temp"`, the full resolution steps for insertion into and retrieval from the RD without any shortcuts are:

B.1.1. Resolving the URIs

The client parses the single returned record. The link's target (sometimes called `"href"`) is `"/sensors/temp"`, which is a relative URI that needs resolving. The base URI `<coap://[ff02::fd]:5683/.well-known/core>` is used to resolve the reference `/sensors/temp` against.

The Base URI of the requested resource can be composed from the options of the CoAP GET request by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into `"coap://[2001:db8:f0::1]/.well-known/core"`.

Because `"/sensors/temp"` starts with a single slash, the record's target is resolved by replacing the path `"/.well-known/core"` from the Base URI (section 5.2 [RFC3986]) with the relative target URI `"/sensors/temp"` into `"coap://[2001:db8:f0::1]/sensors/temp"`.

B.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content format is text/plain (ct=0).

A relation in a web link is a three-part statement that specifies a named relation between the so-called "context resource" and the target resource, like `"_This page_ has _its table of contents_ at _/toc.html_"`. In link format documents, there is an implicit "host relation" specified with default parameter: `rel="hosts"`.

In our example, the context resource of the link is implied to be `"coap://[2001:db8:f0::1]"` by the default value of the anchor (see Appendix B.4). A full English expression of the "host relation" is:

`'"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/sensors/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'`

B.2. A slightly more complex example

Omitting the `"rt=temperature"` filter, the discovery query would have given some more records in the payload:

Req: GET `coap://[ff02::fd]:5683/.well-known/core`

Res: 2.05 Content

Payload:

```
</sensors/temp>;rt=temperature;ct=0,  
</sensors/light>;rt=light-lux;ct=0,  
</t>;anchor="/sensors/temp";rel=alternate,  
<http://www.example.com/sensors/t123>;anchor="/sensors/temp";  
rel=describedby
```

Figure 31: Extended example of direct resource discovery

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the Base URI of the request and is thus resolved to `"coap://[2001:db8:f0::1]/sensors/temp"`. That is the context resource of the link, so the "rel" statement is not about the target and the Base URI any more, but about the target and the resolved URI. Thus, the third record could be read as `"coap://[2001:db8:f0::1]/sensors/temp"` has an alternate representation at `"coap://[2001:db8:f0::1]/t"`.

Following the same resolution steps, the fourth record can be read as `"coap://[2001:db8:f0::1]/sensors/temp"` is described by `"http://www.example.com/sensors/t123"`.

B.3. Enter the Resource Directory

The RD tries to carry the semantics obtainable by classical CoAP discovery over to the resource lookup interface as faithfully as possible.

For the following queries, we will assume that the simple host has used Simple Registration to register at the RD that was announced to it, sending this request from its UDP port `"[2001:db8:f0::1]:6553"`:

Req: POST `coap://[2001:db8:f01::ff]/.well-known/rd?ep=simple-host1`

Res: 2.04 Changed

Figure 32: Example of a simple registration

The RD would have accepted the registration, and queried the simple host's `"/.well-known/core"` by itself. As a result, the host is registered as an endpoint in the RD with the name `"simple-host1"`. The registration is active for 90000 seconds, and the endpoint registration Base URI is `"coap://[2001:db8:f0::1]"` following the resolution steps described in Appendix B.1.1. It should be remarked that the Base URI constructed that way always yields a URI of the form: `scheme://authority without path suffix`.

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching `"coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res"`, obtain `"coap://[2001:db8:f0::ff]/rd-lookup/res"` as the resource lookup endpoint, and ask it for all temperature resources:

Req: GET coap://[2001:db8:f0::ff]/rd-lookup/res?rt=temperature

Res: 2.05 Content

Payload:

<coap://[2001:db8:f0::1]/sensors/temp>;rt=temperature;ct=0

Figure 33: Example exchange performing resource lookup

This is not literally the same response that it would have received from a multicast request, but it contains the equivalent statement:

'"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/sensors/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'

To complete the examples, the client could also query all resources hosted at the endpoint with the known endpoint name "simple-host1":

Req: GET coap://[2001:db8:f0::ff]/rd-lookup/res?ep=simple-host1

Res: 2.05 Content

Payload:

<coap://[2001:db8:f0::1]/sensors/temp>;rt=temperature;ct=0,
 <coap://[2001:db8:f0::1]/sensors/light>;rt=light-lux;ct=0,
 <coap://[2001:db8:f0::1]/t>;
 anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=alternate,
 <http://www.example.com/sensors/t123>;
 anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=describedby

Figure 34: Extended example exchange performing resource lookup

All the target and anchor references are already in absolute form there, which don't need to be resolved any further.

Had the simple host done an equivalent full registration with a base= parameter (e.g. "?ep=simple-host1&base=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

<coap+tcp://simple-host1.example.com/sensors/temp>;rt=temperature;ct=0

Figure 35: Example payload of a response to a resource lookup with a dedicated base URI

and analogous records.

B.4. A note on differences between link-format and Link header fields

While link-format and Link header fields look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC8288]. When implementing an RD or interacting with an RD, care must be taken to follow the [RFC6690] behavior whenever application/link-format representations are used.

- * "Default value of anchor": Both under [RFC6690] and [RFC8288], relative references in the term inside the angle brackets (the target) and the anchor attribute are resolved against the relevant base URI (which usually is the URI used to retrieve the entity), and independent of each other.

When, in an [RFC8288] Link header, the anchor attribute is absent, the link's context is the URI of the selected representation (and usually equal to the base URI).

In [RFC6690] links, if the anchor attribute is absent, the default value is the Origin of (for all relevant cases: the URI reference "/" resolved against) the link's target.

- * There is no percent encoding in link-format documents.

A link-format document is a UTF-8 encoded string of Unicode characters and does not have percent encoding, while Link header fields are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explicitly when required.

For example, while a Link header field in a page about a Swedish city might read

```
Link: </temperature/Malm%C3%B6>;rel=live-environment-data
```

a link-format document from the same source might describe the link as

```
</temperature/Malmö>;rel=live-environment-data
```

Appendix C. Limited Link Format

The CoRE Link Format as described in [RFC6690] has been interpreted differently by implementers, and a strict implementation rules out some use cases of an RD (e.g. base values with path components in combination with absent anchors).

This appendix describes a subset of link format documents called Limited Link Format. The one rule herein is not very limiting in practice -- all examples in RFC6690, and all deployments the authors are aware of already stick to them -- but ease the implementation of RD servers.

It is applicable to representations in the application/link-format media type, and any other media types that inherit [RFC6690] Section 2.1.

A link format representation is in Limited Link format if, for each link in it, the following applies:

All URI references either follow the URI or the path-absolute ABNF rule of RFC3986 (i.e. target and anchor each either start with a scheme or with a single slash).

Authors' Addresses

Christian Amsüss (editor)
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: christian@amsuess.com

Zach Shelby
ARM
150 Rose Orchard
San Jose, 95134
United States of America

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View, 94043
United States of America

Phone: +1-707-502-5136
Email: Michael.Koster@smarththings.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 7, 2016

C. Jennings
Cisco
Z. Shelby
ARM
J. Arkko
A. Keranen
Ericsson
April 5, 2016

Media Types for Sensor Markup Language (SenML)
draft-jennings-core-senml-06

Abstract

This specification defines media types for representing simple sensor measurements and device parameters in the Sensor Markup Language (SenML). Representations are defined in JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR), eXtensible Markup Language (XML), and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use this media type in protocols such as HTTP or CoAP to transport the measurements of the sensor or to be configured.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 7, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Overview	3
2. Requirements and Design Goals	3
3. Terminology	5
4. Semantics	5
5. Associating Meta-data	8
6. JSON Representation (application/senml+json)	8
6.1. Examples	9
6.1.1. Single Datapoint	9
6.1.2. Multiple Datapoints	9
6.1.3. Multiple Measurements	10
6.1.4. Collection of Resources	11
7. CBOR Representation (application/senml+cbor)	12
8. XML Representation (application/senml+xml)	13
9. EXI Representation (application/senml-exi)	15
10. Usage Considerations	19
11. CDDL	20
12. IANA Considerations	21
12.1. Units Registry	21
12.2. SenML label registry	24
12.3. Media Type Registration	24
12.3.1. senml+json Media Type Registration	24
12.3.2. senml+cbor Media Type Registration	25
12.3.3. senml+xml Media Type Registration	26
12.3.4. senml-exi Media Type Registration	27
12.4. XML Namespace Registration	28
12.5. CoAP Content-Format Registration	28
13. Security Considerations	28
14. Privacy Considerations	29
15. Acknowledgement	29
16. References	29
16.1. Normative References	29
16.2. Informative References	30
Appendix A. Links extension	31
Authors' Addresses	32

1. Overview

Connecting sensors to the internet is not new, and there have been many protocols designed to facilitate it. This specification defines new media types for carrying simple sensor information in a protocol such as HTTP or CoAP called the Sensor Markup Language (SenML). This format was designed so that processors with very limited capabilities could easily encode a sensor measurement into the media type, while at the same time a server parsing the data could relatively efficiently collect a large number of sensor measurements. The markup language can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and the web resource model where the sensor is requested as a resource representation (e.g., "GET /sensor/temperature").

There are many types of more complex measurements and measurements that this media type would not be suitable for. SenML strikes a balance between having some information about the sensor carried with the sensor data so that the data is self describing but it also tries to make that a fairly minimal set of auxiliary information for efficiency reason. Other information about the sensor can be discovered by other methods such as using the CoRE Link Format [RFC6690].

SenML is defined by a data model for measurements and simple meta-data about measurements and devices. The data is structured as a single array that contains a series of SenML Records which can each contain attributes such as an unique identifier for the sensor, the time the measurement was made, the unit the measurement is in, and the current value of the sensor. Serializations for this data model are defined for JSON [RFC7159], CBOR [RFC7049], XML, and Efficient XML Interchange (EXI) [W3C.REC-exi-20110310].

For example, the following shows a measurement from a temperature gauge encoded in the JSON syntax.

```
[{ "n": "urn:dev:ow:10e2073a01080063", "v":23.1, "u":"Cel" }]
```

In the example above, the array has a single SenML Record with a measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a current value of 23.5 degrees Celsius.

2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets on mesh networks from large numbers of constrained devices. Keeping the total size of payload under 80 bytes makes this easy to use on a wireless mesh network. It is always difficult to

define what small code is, but there is a desire to be able to implement this in roughly 1 KB of flash on a 8 bit microprocessor. Experience with Google power meter and large scale deployments has indicated that the solution needs to support allowing multiple measurements to be batched into a single HTTP or CoAP request. This "batch" upload capability allows the server side to efficiently support a large number of devices. It also conveniently supports batch transfers from proxies and storage devices, even in situations where the sensor itself sends just a single data item at a time. The multiple measurements could be from multiple related sensors or from the same sensor but at different times.

The basic design is an array with a series of measurements. The following example shows two measurements made at different times. The value of a measurement is in the "v" tag, the time of a measurement is in the "t" tag, the "n" tag has a unique sensor name, and the unit of the measurement is carried in the "u" tag.

```
[
  { "n": "urn:dev:ow:10e2073a01080063",
    "t": 1276020076, "v":23.5, "u":"Cel" },
  { "n": "urn:dev:ow:10e2073a01080063",
    "t": 1276020091, "v":23.6, "u":"Cel" }
]
```

To keep the messages small, it does not make sense to repeat the "n" tag in each SenML Record so there is a concept of a Base Name which is simply a string that is prepended to the Name field of all elements in that record and any records that follow it. So a more compact form of the example above is the following.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063",
    "t": 1276020076, "v":23.5, "u":"Cel" },
  { "t": 1276020091, "v":23.6, "u":"Cel" }
]
```

In the above example the Base Name is in the "bn" tag and the "n" tags in each Record are the empty string so they are omitted. The Base Name also could be put in a separate Record such as in the following example.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063" },
  { "t": 1276020076, "v":23.5, "u":"Cel" },
  { "t": 1276020091, "v":23.6, "u":"Cel" }
]
```

Some devices have accurate time while others do not so SenML supports absolute and relative times. Time is represented in floating point as seconds and values greater than zero represent an absolute time relative to the unix epoch while values of 0 or less represent a relative time in the past from the current time. A simple sensor with no absolute wall clock time might take a measurement every second and batch up 60 of them then send it to a server. It would include the relative time the measurement was made to the time the batch was send in the SenML Pack. The server might have accurate NTP time and use the time it received the data, and the relative offset, to replace the times in the SenML with absolute times before saving the SenML Pack in a document database.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document also uses the following terms:

SenML Record: One measurement or configuration instance in time presented using the SenML data model.

SenML Pack: One or more SenML Records in an array structure.

4. Semantics

Each SenML Pack carries a single array that represents a set of measurements and/or parameters. This array contains a series of objects with several optional attributes described below:

Base Name: This is a string that is prepended to the names found in the entries. This attribute is optional. This applies to the entries in all Records. A Base Name can only be included in the first Record of the array.

Base Time: A base time that is added to the time found in an entry. This attribute is optional. This applies to the entries in all Records. A Base Time can only be included in the first Record of the array.

Base Unit: A base unit that is assumed for all entries, unless otherwise indicated. This attribute is optional. If a record does not contain a unit value, then the base unit is used otherwise the value of found in the Unit is used. This applies to

the entries in all Records. A Base Unit can only be included in the first object of the array.

Base Value: A base value is added to the value found in an entry, similar to Base Time. This attribute is optional. This applies to the entries in all Records. A Base Value can only be included in the first Record of the array.

Version: Version number of media type format. This attribute is optional positive integer and defaults to 5 if not present. A Version can only be included in the first object of the array.

Name: Name of the sensor or parameter. When appended to the Base Name attribute, this must result in a globally unique identifier for the resource. The name is optional, if the Base Name is present. If the name is missing, Base Name must uniquely identify the resource. This can be used to represent a large array of measurements from the same sensor without having to repeat its identifier on every measurement.

Unit: Units for a measurement value. Optional. If the Record has not Unit, the Base Unit is used as the Unit. Having no Unit and no Base Unit is allowed.

Value Value of the entry. Optional if a Sum value is present, otherwise required. Values are represented using three basic data types, Floating point numbers ("v" field for "Value"), Booleans ("vb" for "Boolean Value"), Strings ("vs" for "String Value") and Data ("vd" for "Binary Data Value"). Exactly one of these three fields MUST appear unless there is Sum field in which case it is allowed to have no Value field or to have "v" field.

Sum: Integrated sum of the values over time. Optional. This attribute is in the units specified in the Unit value multiplied by seconds.

Time: Time when value was recorded. Optional.

Update Time: An optional time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. This can be used to detect the failure of sensors or communications path from the sensor.

The SenML format can be extended with further custom attributes. TODO - describe what extensions are possible and how to do them.

Systems reading one of the objects MUST check for the Version attribute. If this value is a version number larger than the version

which the system understands, the system SHOULD NOT use this object. This allows the version number to indicate that the object contains mandatory to understand attributes. New version numbers can only be defined in an RFC that updates this specification or its successors.

The Name value is concatenated to the Base Name value to get the name of the sensor. The resulting name needs to uniquely identify and differentiate the sensor from all others. If the object is a representation resulting from the request of a URI [RFC3986], then in the absence of the Base Name attribute, this URI is used as the default value of Base Name. Thus in this case the Name field needs to be unique for that URI, for example an index or subresource name of sensors handled by the URI.

Alternatively, for objects not related to a URI, a unique name is required. In any case, it is RECOMMENDED that the full names are represented as URIs or URNs [RFC2141]. One way to create a unique name is to include some bit string that has guaranteed uniqueness (such as a 1-wire address) that is assigned to the device. Some of the examples in this draft use the device URN type as specified in [I-D.arkko-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name. Note that long-term stable unique identifiers are problematic for privacy reasons [RFC7721] and should be used with care or avoided.

The resulting concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", "-", ":", ".", or "_" and it MUST start with a character out of the set "A" to "Z", "a" to "z", or "0" to "9". This restricted character set was chosen so that these names can be directly used as in other types of URI including segments of an HTTP path with no special encoding and can be directly used in many databases and analytic systems. [RFC5952] contains advice on encoding an IPv6 address in a name.

If either the Base Time or Time value is missing, the missing attribute is considered to have a value of zero. The Base Time and Time values are added together to get the time of measurement. A time of zero indicates that the sensor does not know the absolute time and the measurement was made roughly "now". A negative value is used to indicate seconds in the past from roughly "now". A positive value is used to indicate the number of seconds, excluding leap seconds, since the start of the year 1970 in UTC.

Representing the statistical characteristics of measurements, such as accuracy, can be very complex. Future specification may add new attributes to provide better information about the statistical properties of the measurement.

5. Associating Meta-data

SenML is designed to carry the minimum dynamic information about measurements, and for efficiency reasons does not carry significant static meta-data about the device, object or sensors. Instead, it is assumed that this meta-data is carried out of band. For web resources using SenML Packs, this meta-data can be made available using the CoRE Link Format [RFC6690]. The most obvious use of this link format is to describe that a resource is available in a SenML format in the first place. The relevant media type indicator is included in the Content-Type (ct=) attribute.

6. JSON Representation (application/senml+json)

Record attributes:

SenML	JSON	Type
Base Name	bn	String
Base Time	bt	Number
Base Unit	bu	String
Base Value	bv	Number
Version	bver	Number
Name	n	String
Unit	u	String
Value	v	Number
String Value	vs	String
Boolean Value	vb	Boolean
Data Value	vd	String
Value Sum	s	Number
Time	t	Number
Update Time	ut	Number

The root content consists of an array with JSON objects for each SenML Record. All the fields in the above table MAY occur in the records with the type specified in the table.

Only the UTF-8 form of JSON is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC4627]. Characters in the Data Value are base64 encoded with URL safe alphabet as defined in Section 5 of [RFC4648].

Systems receiving measurements MUST be able to process the range of floating point numbers that are representable as an IEEE double-precision floating-point numbers [IEEE.754.1985]. The number of significant digits in any measurement is not relevant, so a reading

of 1.1 has exactly the same semantic meaning as 1.10. If the value has an exponent, the "e" MUST be in lower case. The mantissa SHOULD be less than 19 characters long and the exponent SHOULD be less than 5 characters long. This allows time values to have better than micro second precision over the next 100 years.

6.1. Examples

TODO - simplify examples

TODO - Examples are messed up on if time is an integer or float

TODO - Add example with string, data, boolean, and base value

6.1.1. Single Datapoint

The following shows a temperature reading taken approximately "now" by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

```
[{ "n": "urn:dev:ow:10e2073a01080063", "v":23.1, "u":"Cel" }]
```

6.1.2. Multiple Datapoints

The following example shows voltage and current now, i.e., at an unspecified time.

```
[{"bn": "urn:dev:ow:10e2073a01080063/"},
 { "n": "voltage", "t": 0, "u": "V", "v": 120.1 },
 { "n": "current", "t": 0, "u": "A", "v": 1.2 }
]
```

The next example is similar to the above one, but shows current at Tue Jun 8 18:01:16 UTC 2010 and at each second for the previous 5 seconds.

```
[{"bn": "urn:dev:ow:10e2073a01080063/",
 "bt": 1276020076.001,
 "bu": "A",
 "bver": 5},
 { "n": "voltage", "u": "V", "v": 120.1 },
 { "n": "current", "t": -5, "v": 1.2 },
 { "n": "current", "t": -4, "v": 1.30 },
 { "n": "current", "t": -3, "v": 0.14e1 },
 { "n": "current", "t": -2, "v": 1.5 },
 { "n": "current", "t": -1, "v": 1.6 },
 { "n": "current", "t": 0, "v": 1.7 }
]
```

Note that in some usage scenarios of SenML the implementations MAY store or transmit SenML in a stream-like fashion, where data is collected over time and continuously added to the object. This mode of operation is optional, but systems or protocols using SenML in this fashion MUST specify that they are doing this. SenML defines a separate mime type (TODO) to indicate Sensor Streaming Markup Language (SensML) for this usage. In this situation the SensML stream can be sent and received in a partial fashion, i.e., a measurement entry can be read as soon as the SenML Record is received and not have to wait for the full SensML Stream to be complete.

For instance, the following stream of measurements may be sent via a long lived HTTP POST from the producer of a SensML to the consumer of that, and each measurement object may be reported at the time it measured:

```
[ { "bn": "urn:dev:ow:10e2073a01080063",  
  "bt": 1320067464,  
  "bu": "%RH"},  
  { "v": 21.2, "t": 0 },  
  { "v": 21.3, "t": 10 },  
  { "v": 21.4, "t": 20 },  
  { "v": 21.4, "t": 30 },  
  { "v": 21.5, "t": 40 },  
  { "v": 21.5, "t": 50 },  
  { "v": 21.5, "t": 60 },  
  { "v": 21.6, "t": 70 },  
  { "v": 21.7, "t": 80 },  
  { "v": 21.5, "t": 90 },  
  ...
```

6.1.3. Multiple Measurements

The following example shows humidity measurements from a mobile device with an IPv6 address 2001:db8::1, starting at Mon Oct 31 13:24:24 UTC 2011. The device also provides position data, which is provided in the same measurement or parameter array as separate entries. Note time is used to for correlating data that belongs together, e.g., a measurement and a parameter associated with it. Finally, the device also reports extra data about its battery status at a separate time.

```
[{"bn": "urn:dev:ow:10e2073a01080063",
  "bt": 1320067464,
  "bu": "%RH"},
 { "v": 20.0, "t": 0 },
 { "v": 24.30621, "u": "lon", "t": 0 },
 { "v": 60.07965, "u": "lat", "t": 0 },
 { "v": 20.3, "t": 60 },
 { "v": 24.30622, "u": "lon", "t": 60 },
 { "v": 60.07965, "u": "lat", "t": 60 },
 { "v": 20.7, "t": 120 },
 { "v": 24.30623, "u": "lon", "t": 120 },
 { "v": 60.07966, "u": "lat", "t": 120 },
 { "v": 98.0, "u": "%EL", "t": 150 },
 { "v": 21.2, "t": 180 },
 { "v": 24.30628, "u": "lon", "t": 180 },
 { "v": 60.07967, "u": "lat", "t": 180 }
]
```

The size of this example represented in various forms, as well as that form compressed with gzip is given in the following table.

Encoding	Size	Compressed Size
JSON	567	200
XML	656	232
CBOR	292	192
EXI	160	183

Table 1: Size Comparisons

Note the CBOR and EXI sizes are not using the schema guidance so the could be a bit smaller.

6.1.4. Collection of Resources

The following example shows how to query one device that can provide multiple measurements. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on `http://[2001:db8::2]` at Mon Oct 31 16:27:09 UTC 2011, and has gotten two separate values as a result, a temperature and humidity measurement.

```
[{"bn": "urn:dev:ow:10e2073a01080063/",
  "bt": 1320078429,
  "bver": 5
},
{ "n": "temperature", "v": 27.2, "u": "Cel" },
{ "n": "humidity", "v": 80, "u": "%RH" }
]
```

7. CBOR Representation (application/senml+cbor)

The CBOR [RFC7049] representation is equivalent to the JSON representation, with the following changes:

- o For compactness, the CBOR representation uses integers for the map keys defined in Table 2. This table is conclusive, i.e., there is no intention to define any additional integer map keys; any extensions will use string map keys.
- o For JSON Numbers, the CBOR representation can use integers, floating point numbers, or decimal fractions (CBOR Tag 4); the common limitations of JSON implementations are not relevant for these. For the version number, however, only an unsigned integer is allowed.

Name	JSON label	CBOR label
Version	bver	-1
Base Name	bn	-2
Base Time	bt	-3
Base Units	bu	-4
Base Value	bv	-5
Name	n	0
Units	u	1
Value	v	2
String Value	vs	3
Boolean Value	vb	4
Value Sum	s	5
Time	t	6
Update Time	ut	7
Data Value	vd	8

Table 2: CBOR representation: integers for map keys

The following example shows an hexdump of the CBOR example for the same sensor measurement as in Section 6.1.2.

0000 88 a4 62 62 6e 78 1c 75 72 6e 3a 64 65 76 3a 6f	..bbnx.urn:dev:ow:10e2073a01080063/bbt.L..lbbuaA
0010 77 3a 31 30 65 32 30 37 33 61 30 31 30 38 30 30	w:10e2073a01080063/bbt.L..lbbuaA
0020 36 33 2f 62 62 74 1a 4c 0e 85 6c 62 62 75 61 41	63/bbt.L..lbbuaA
0030 63 76 65 72 05 a3 61 6e 67 76 6f 6c 74 61 67 65	cver..angvoltage
0040 61 75 61 56 61 76 fb 40 5e 06 66 66 66 66 66 a3	auaVav.@^.ffffff.
0050 61 6e 67 63 75 72 72 65 6e 74 61 74 24 61 76 fb	angcurrentat\$av.
0060 3f f3 33 33 33 33 33 33 a3 61 6e 67 63 75 72 72	?.333333.angcurr
0070 65 6e 74 61 74 23 61 76 fb 3f f4 cc cc cc cc cc	entat#av.?.....
0080 cd a3 61 6e 67 63 75 72 72 65 6e 74 61 74 22 61	..angcurrentat"a
0090 76 fb 3f f6 66 66 66 66 66 66 66 a3 61 6e 67 63 75	v.?.ffffff.angcu
00a0 72 72 65 6e 74 61 74 21 61 76 fb 3f f8 00 00 00	rrentat!av.?....
00b0 00 00 00 a3 61 6e 67 63 75 72 72 65 6e 74 61 74angcurrentat
00c0 20 61 76 fb 3f f9 99 99 99 99 99 9a a2 61 6e 67	av.?.....ang
00d0 63 75 72 72 65 6e 74 61 76 fb 3f fb 33 33 33 33	currentav.?.3333
00e0 33 33 0a	33.
00e3	

8. XML Representation (application/senml+xml)

A SenML Stream can also be represented in XML format as defined in this section. The following example shows an XML example for the same sensor measurement as in Section 6.1.2.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a01080063/" bt="1.276020076e+09"
    bu="A" bver="5"></senml>
  <senml n="voltage" u="V" v="120.1"></senml>
  <senml n="current" t="-5" v="1.2"></senml>
  <senml n="current" t="-4" v="1.3"></senml>
  <senml n="current" t="-3" v="1.4"></senml>
  <senml n="current" t="-2" v="1.5"></senml>
  <senml n="current" t="-1" v="1.6"></senml>
  <senml n="current" v="1.7"></senml>
</sensml>
```

The SenML Stream is represented as a `sensml` tag that contains a series of `senml` tags for each SenML Record. The SenML Fields are represented as XML attributes. The following table shows the mapping the SenML Field names to the attribute used in the XML `senml` tag.

SenML Field	XML	Type
Base Name	bn	string
Base Time	bt	float
Base Unit	bu	string
Base Value	bv	float
Version	bver	int
Name	n	string
Unit	u	string
Value	v	float
String Value	vs	string
Data Value	vd	string
Boolean Value	vb	boolean
Value Sum	s	float
Time	t	float
Update Time	ut	float

The RelaxNG schema for the XML is:

```
default namespace = "urn:ietf:params:xml:ns:senml"
namespace rng = "http://relaxng.org/ns/structure/1.0"

link = element l {
    attribute * { xsd:string }*
}

senml = element senml {
    attribute bn { xsd:string }?,
    attribute bt { xsd:double }?,
    attribute bv { xsd:double }?,
    attribute bu { xsd:string }?,
    attribute bver { xsd:int }?,

    attribute n { xsd:string }?,
    attribute s { xsd:double }?,
    attribute t { xsd:double }?,
    attribute u { xsd:string }?,
    attribute ut { xsd:double }?,

    attribute v { xsd:double }?,
    attribute vb { xsd:boolean }?,
    attribute vs { xsd:string }?,
    attribute vd { xsd:string }?,

    link*
}

sensml =
    element sensml {
        senml+
    }

start = sensml
```

9. EXI Representation (application/senml-exi)

For efficient transmission of SenML over e.g. a constrained network, Efficient XML Interchange (EXI) can be used. This encodes the XML Schema structure of SenML into binary tags and values rather than ASCII text. An EXI representation of SenML SHOULD be made using the strict schema-mode of EXI. This mode however does not allow tag extensions to the schema, and therefore any extensions will be lost in the encoding. For uses where extensions need to be preserved in EXI, the non-strict schema mode of EXI MAY be used.

The EXI header option MUST be included. An EXI schemaID options MUST be set to the value of "a" indicating the scheme provided in this specification. Future revisions to the schema can change this schemaID to allow for backwards compatibility. When the data will be transported over CoAP or HTTP, an EXI Cookie SHOULD NOT be used as it simply makes things larger and is redundant to information provided in the Content-Type header.

TODO - examples probably have the wrong setting the schemaID

The following is the XSD Schema to be used for strict schema guided EXI processing. It is generated from the RelaxNG.


```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="urn:ietf:params:xml:ns:senml"
  xmlns:ns1="urn:ietf:params:xml:ns:senml">
  <xs:element name="l">
    <xs:complexType>
      <xs:anyAttribute processContents="skip" />
    </xs:complexType>
  </xs:element>
  <xs:element name="senml">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded"
          ref="ns1:l" />
      </xs:sequence>
      <xs:attribute name="bn" type="xs:string" />
      <xs:attribute name="bt" type="xs:double" />
      <xs:attribute name="bv" type="xs:double" />
      <xs:attribute name="bu" type="xs:string" />
      <xs:attribute name="bver" type="xs:int" />
      <xs:attribute name="n" type="xs:string" />
      <xs:attribute name="s" type="xs:double" />
      <xs:attribute name="t" type="xs:double" />
      <xs:attribute name="u" type="xs:string" />
      <xs:attribute name="ut" type="xs:double" />
      <xs:attribute name="v" type="xs:double" />
      <xs:attribute name="vb" type="xs:boolean" />
      <xs:attribute name="vs" type="xs:string" />
      <xs:attribute name="vd" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="sensml">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="ns1:senml" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The following shows a hexdump of the EXI produced from encoding the following XML example. Note this example is the same information as the first example in Section 6.1.2 in JSON format.

```
<sensml xmlns="urn:ietf:params:xml:ns:sensml">
  <senml bn="urn:dev:ow:10e2073a01080063/"></senml>
  <senml n="voltage" u="V" v="120.1"></senml>
  <senml n="current" u="A" v="1.2"></senml>
</sensml>
```

Which compresses with EXI to the following displayed in hexdump:

```
0000 a0 30 41 cd 95 b9 b5 b0 d4 b9 9d 95 b8 b9 e1 cd |.0A.....|
0010 91 00 f3 ab 93 71 d3 23 2b b1 d3 7b b9 d1 89 83 |.....q.#+...{|
0020 29 91 81 b9 9b 09 81 89 81 c1 81 81 b1 99 7f 14 |).....|
0030 25 d9 bd b1 d1 85 9d 94 80 d5 8a c4 26 01 0a 12 |%......&...|
0040 c6 ea e4 e4 ca dc e8 40 68 24 19 00 90          |.....@h$....|
004d
```

The above example used the bit packed form of EXI but it is also possible to use a byte packed form of EXI which can makes it easier for a simple sensor to produce valid EXI without really implementing EXI. Consider the example of a temperature sensor that produces a value in tenths of degrees Celsius over a range of 0.0 to 55.0. It would produce an XML SenML file such as:

```
<sensml xmlns="urn:ietf:params:xml:ns:sensml">
  <senml n="urn:dev:ow:10e2073a01080063" u="Cel" v="23.1"></senml>
</sensml>
```

The compressed form, using the byte alignment option of EXI, for the above XML is the following:

```
0000 a0 00 48 82 0e 6c ad cd ad 86 a5 cc ec ad c5 cf |..H..l.....|
0010 0e 6c 80 02 05 1d 75 72 6e 3a 64 65 76 3a 6f 77 |.l....urn:dev:ow|
0020 3a 31 30 65 32 30 37 33 61 30 31 30 38 30 30 36 |:10e2073a0108006|
0030 33 02 05 43 65 6c 01 00 e7 01 01 00 04 01      |3..Cel.....|
003e
```

A small temperature sensor devices that only generates this one EXI file does not really need an full EXI implementation. It can simple hard code the output replacing the one wire device ID starting at byte 0x16 and going to byte 0x31 with it's device ID, and replacing the value "0xe7 0x01" at location 0x38 to 0x39 with the current temperature. The EXI Specification [W3C.REC-exi-20110310] contains the full information 'on how floating point numbers are represented, but for the purpose of this sensor, the temperature can be converted to an integer in tenths of degrees (231 in this example). EXI stores 7 bits of the integer in each byte with the top bit set to one if there are further bytes. So the first bytes at is set to low 7 bits of the integer temperature in tenths of degrees plus 0x80. In this example $231 \& 0x7F + 0x80 = 0xE7$. The second byte is set to the

integer temperature in tenths of degrees right shifted 7 bits. In this example $231 \gg 7 = 0x01$.

10. Usage Considerations

The measurements support sending both the current value of a sensor as well as the an integrated sum. For many types of measurements, the sum is more useful than the current value. For example, an electrical meter that measures the energy a given computer uses will typically want to measure the cumulative amount of energy used. This is less prone to error than reporting the power each second and trying to have something on the server side sum together all the power measurements. If the network between the sensor and the meter goes down over some period of time, when it comes back up, the cumulative sum helps reflect what happened while the network was down. A meter like this would typically report a measurement with the units set to watts, but it would put the sum of energy used in the "s" attribute of the measurement. It might optionally include the current power in the "v" attribute.

While the benefit of using the integrated sum is fairly clear for measurements like power and energy, it is less obvious for something like temperature. Reporting the sum of the temperature makes it easy to compute averages even when the individual temperature values are not reported frequently enough to compute accurate averages. Implementors are encouraged to report the cumulative sum as well as the raw value of a given sensor.

Applications that use the cumulative sum values need to understand they are very loosely defined by this specification, and depending on the particular sensor implementation may behave in unexpected ways. Applications should be able to deal with the following issues:

1. Many sensors will allow the cumulative sums to "wrap" back to zero after the value gets sufficiently large.
2. Some sensors will reset the cumulative sum back to zero when the device is reset, loses power, or is replaced with a different sensor.
3. Applications cannot make assumptions about when the device started accumulating values into the sum.

Typically applications can make some assumptions about specific sensors that will allow them to deal with these problems. A common assumption is that for sensors whose measurement values are always positive, the sum should never get smaller; so if the sum does get

smaller, the application will know that one of the situations listed above has happened.

11. CDDL

For reference, the CBOR representation can be described with the CDDL [I-D.greevenbosch-appsawg-cbor-cddl] specification in Figure 1.

```
SenML-Pack = [initial-record, * follow-on-record]
```

```
initial-record = initial-defined .and initial-generic
```

```
follow-on-record = follow-on-defined .and follow-on-generic
```

```
; first do a specification of the labels as defined:
```

```
initial-defined = {
  ? bn => tstr,          ; Base Name
  ? bt => numeric,       ; Base Time
  ? bu => tstr,          ; Base Units
  ? bv => numeric,       ; Base value
  ? bver => uint,        ; Base Version
  follow-on-defined-group,
  + base-key-value-pair
}
```

```
follow-on-defined-group = (
  ? n => tstr,          ; Name
  ? u => tstr,          ; Units
  ? ( v => numeric // ; Numeric Value
    vs => tstr // ; String Value
    vb => bool // ; Boolean Value
    vd => bstr ) ; Data Value
  ? s => numeric,       ; Value Sum
  ? t => numeric,       ; Time
  ? ut => numeric,      ; Update Time
  * key-value-pair
)
```

```
follow-on-defined = { follow-on-defined-group }
```

```
; CBOR version (use the labels)
```

```
bver = -1  n  = 0  s  = 5
```

```
bn  = -2   u  = 1   t  = 6
```

```
bt  = -3   v  = 2   ut = 7
```

```
bu  = -4   vs = 3   vd = 8
```

```
bv  = -5   vb = 4
```

```
; use the label *names* for JSON
```

```
; now define the generic versions
```

```

initial-generic = {
    follow-on-generic-group,
    * base-key-value-pair,
}

follow-on-generic-group = (
    + key-value-pair,
)
follow-on-generic = { follow-on-generic-group }

key-value-pair = ( non-b-label => value )

base-key-value-pair = ( b-label => value )

non-b-label = tstr .regex "[A-Za-z0-9][_:.A-Za-z0-9]*" / uint
b-label = tstr .regex "b[_:.A-Za-z0-9]+" / nint

value = tstr / bstr / numeric / bool
numeric = number / decfrac

```

Figure 1: CDDL specification for CBOR SenML

12. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification.

12.1. Units Registry

IANA will create a registry of SenML unit symbols. The primary purpose of this registry is to make sure that symbols uniquely map to give type of measurement. Definitions for many of these units can be found in location such as [NIST811] and [BIPM].

Symbol	Description	Type	Reference
m	meter	float	RFC-AAAA
g	gram	float	RFC-AAAA
s	second	float	RFC-AAAA
A	ampere	float	RFC-AAAA
K	kelvin	float	RFC-AAAA
cd	candela	float	RFC-AAAA
mol	mole	float	RFC-AAAA
Hz	hertz	float	RFC-AAAA
rad	radian	float	RFC-AAAA
sr	steradian	float	RFC-AAAA

N	newton	float	RFC-AAAA
Pa	pascal	float	RFC-AAAA
J	joule	float	RFC-AAAA
W	watt	float	RFC-AAAA
C	coulomb	float	RFC-AAAA
V	volt	float	RFC-AAAA
F	farad	float	RFC-AAAA
Ohm	ohm	float	RFC-AAAA
S	siemens	float	RFC-AAAA
Wb	weber	float	RFC-AAAA
T	tesla	float	RFC-AAAA
H	henry	float	RFC-AAAA
Cel	degrees Celsius	float	RFC-AAAA
lm	lumen	float	RFC-AAAA
lx	lux	float	RFC-AAAA
Bq	becquerel	float	RFC-AAAA
Gy	gray	float	RFC-AAAA
Sv	sievert	float	RFC-AAAA
kat	katal	float	RFC-AAAA
pH	pH acidity	float	RFC-AAAA
%	Value of a switch (note 1)	float	RFC-AAAA
count	counter value	float	RFC-AAAA
%RH	Relative Humidity	float	RFC-AAAA
m2	area	float	RFC-AAAA
l	volume in liters	float	RFC-AAAA
m/s	velocity	float	RFC-AAAA
m/s2	acceleration	float	RFC-AAAA
l/s	flow rate in liters per second	float	RFC-AAAA
W/m2	irradiance	float	RFC-AAAA
cd/m2	luminance	float	RFC-AAAA
Bspl	bel sound pressure level	float	RFC-AAAA
bit/s	bits per second	float	RFC-AAAA
lat	degrees latitude (note 2)	float	RFC-AAAA
lon	degrees longitude (note 2)	float	RFC-AAAA
%EL	remaining battery energy level in percents	float	RFC-AAAA
EL	remaining battery energy level in seconds	float	RFC-AAAA
beat/m	Heart rate in beats per minute	float	RFC-AAAA
beats	Cumulative number of heart beats	float	RFC-AAAA

Table 3

- o Note 1: A value of 0.0 indicates the switch is off while 1.0 indicates on and 0.5 would be half on.

- o Note 2: Assumed to be in WGS84 unless another reference frame is known for the sensor.

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider the following guidelines:

1. There needs to be a real and compelling use for any new unit to be added.
2. Units should define the semantic information and be chosen carefully. Implementors need to remember that the same word may be used in different real-life contexts. For example, degrees when measuring latitude have no semantic relation to degrees when measuring temperature; thus two different units are needed.
3. These measurements are produced by computers for consumption by computers. The principle is that conversion has to be easily be done when both reading and writing the media type. The value of a single canonical representation outweighs the convenience of easy human representations or loss of precision in a conversion.
4. Use of SI prefixes such as "k" before the unit is not allowed. Instead one can represent the value using scientific notation such a 1.2e3. TODO - Open Issue. Some people would like to have SI prefixes to improve human readability.
5. For a given type of measurement, there will only be one unit type defined. So for length, meters are defined and other lengths such as mile, foot, light year are not allowed. For most cases, the SI unit is preferred.
6. Symbol names that could be easily confused with existing common units or units combined with prefixes should be avoided. For example, selecting a unit name of "mph" to indicate something that had nothing to do with velocity would be a bad choice, as "mph" is commonly used to mean miles per hour.
7. The following should not be used because the are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, n, u, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi.
8. The following units should not be used as they are commonly used to represent other measurements Ky, Gal, dyn, etg, P, St, Mx, G, Oe, Gb, sb, Lmb, ph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, kWh.

9. The unit names are case sensitive and the correct case needs to be used, but symbols that differ only in case should not be allocated.
10. A number after a unit typically indicates the previous unit raised to that power, and the / indicates that the units that follow are the reciprocal. A unit should have only one / in the name.
11. A good list of common units can be found in the Unified Code for Units of Measure [UCUM].

12.2. SenML label registry

IANA will create a registry for SenML labels. The initial content of the registry are shown in TODO.

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider that shorter labels should have more strict review.

12.3. Media Type Registration

The following registrations are done following the procedure specified in [RFC6838] and [RFC7303].

12.3.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json and sensml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC7159]. See RFC-AAAA for details. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and

instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any JSON key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "ver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.2. senml+cbor Media Type Registration

Type name: application

Subtype name: senml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.3. senml+xml Media Type Registration

Type name: application

Subtype name: senml+xml and sensml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.4. senml-exi Media Type Registration

Type name: application

Subtype name: senml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.4. XML Namespace Registration

This document registers the following XML namespaces in the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:senml

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces

12.5. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the SenML media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. All IDs are assigned from the "Expert Review" (0-255) range. The assigned IDs are show in Table 4.

Media type	ID
application/senml+json	TBD
application/sensml+json	TBD
application/senml+cbor	TBD
application/senml+xml	TBD
application/sensml+xml	TBD
application/senml-exi	TBD

Table 4: CoAP Content-Format IDs

13. Security Considerations

See Section 14. Further discussion of security properties can be found in Section 12.3.

14. Privacy Considerations

Sensor data can range from information with almost no security considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transport protocol such as S/MIME or HTTP with TLS that can provide integrity, confidentiality, and authentication information about the source of the data.

15. Acknowledgement

We would like to thank Lisa Dusseault, Joe Hildebrand, Lyndsay Campbell, Martin Thomson, John Klensin, Bjoern Hoehrmann, Carsten Bormann, and Christian Amsuess for their review comments.

The CBOR Representation text and CDDL was contributed by Carsten Bormann.

16. References

16.1. Normative References

- [BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI)", 8th edition, 2006.
- [IEEE.754.1985] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE Standard 754, August 1985.
- [NIST811] Thompson, A. and B. Taylor, "Guide for the Use of the International System of Units (SI)", NIST Special Publication 811, 2008.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <<http://www.rfc-editor.org/info/rfc4627>>.

- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<http://www.rfc-editor.org/info/rfc6838>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <<http://www.rfc-editor.org/info/rfc7303>>.
- [W3C.REC-exi-20110310]
Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <<http://www.w3.org/TR/2011/REC-exi-20110310>>.

16.2. Informative References

- [I-D.arkko-core-dev-urn]
Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-03 (work in progress), July 2012.

- [I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-08 (work in progress), March 2016.
- [I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-04 (work in progress), November 2015.
- [RFC2141] Moats, R., "URN Syntax", RFC 2141, DOI 10.17487/RFC2141, May 1997, <<http://www.rfc-editor.org/info/rfc2141>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<http://www.rfc-editor.org/info/rfc4122>>.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <<http://www.rfc-editor.org/info/rfc5952>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <<http://www.rfc-editor.org/info/rfc7721>>.
- [UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics, 2013, <<http://unitsofmeasure.org/ucum.html>>.

Appendix A. Links extension

An extension to SenML to support links is expected to be registered and defined by [I-D.ietf-core-links-json].

The link extension can be an array of objects that can be used for additional information. Each object in the Link array is constrained to being a map of strings to strings with unique keys.

The following shows an example of the links extension.

```
[{"bn": "urn:dev:ow:10e2073a01080063/",
  "bt": 1320078429,
  "l": "[{\"href\":\"humidity\", \"foo\":\"bar1\"}]",
  { "n": "temperature", "v": 27.2, "u": "Cel" },
  { "n": "humidity", "v": 80, "u": "%RH" }
]
```

Authors' Addresses

Cullen Jennings
Cisco
400 3rd Avenue SW
Calgary, AB T2P 4H2
Canada

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 8, 2016

M. Koster
ARM Limited
A. Keranen
J. Jimenez
Ericsson
November 5, 2015

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-koster-core-coap-pubsub-04

Abstract

The Constrained Application Protocol (CoAP), and related extensions are intended to support machine-to-machine communication in systems where one or more nodes are resource constrained, in particular for low power wireless sensor networks. This document defines a publish-subscribe broker for CoAP that extends the capabilities of CoAP for supporting nodes with long breaks in connectivity and/or up-time.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 8, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Architecture	4
3.1. CoAP pub/sub Architecture	4
3.2. CoAP pub/sub Broker	4
3.3. CoAP pub/sub Client	5
3.4. CoAP pub/sub Topic	5
4. CoAP pub/sub Function Set	5
4.1. DISCOVER	5
4.2. CREATE	7
4.3. PUBLISH	9
4.4. SUBSCRIBE	11
4.5. UNSUBSCRIBE	13
4.6. READ	14
4.7. REMOVE	15
5. CoAP pub/sub Operation with Resource Directory	16
6. Sleep-Wake Operation	17
7. Simple Flow Control	17
8. Security Considerations	18
9. IANA Considerations	19
9.1. Resource Type value 'core.ps'	19
9.2. Response Code value '2.04'	19
9.3. Response Code value '4.29'	19
10. Acknowledgements	19
11. References	20
11.1. Normative References	20
11.2. Informative References	20
Authors' Addresses	21

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports machine-to-machine communication across networks of constrained devices. CoAP uses a request/response model where clients make requests to servers in order to request actions on resources. Depending on the situation the same device may act either as a server or a client.

One important class of constrained devices includes devices that are intended to run for years from a small battery, or by scavenging energy from their environment. These devices have limited reachability because they spend most of their time in a sleeping

state with no network connectivity. Devices may also have limited reachability due to certain middle-boxes, such as Network Address Translators (NATs) or firewalls. Such middle-boxes often prevent connecting to a device from the Internet unless the connection was initiated by the device.

This document specifies the means for nodes with limited reachability to communicate using simple extensions to CoAP. The extensions enable publish-subscribe communication using a broker node that enables store-and-forward messaging between two or more nodes. Furthermore the extensions facilitate many-to-many communication using CoAP.

2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [RFC2119].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format [RFC6570] is used to describe the REST interfaces defined in this specification.

This specification makes use of the following additional terminology:

Publish-Subscribe (pub/sub): A messaging paradigm where messages are published to a broker and potential receivers can subscribe to the broker to receive messages. The publishers do not (need to) know where the message will be eventually sent: the publications and subscriptions are matched by a broker and publications are delivered by the broker to subscribed receivers.

CoAP pub/sub function set: A group of well-known REST resources that together provide the CoAP pub/sub service.

CoAP pub/sub Broker: A server node capable of receiving messages (publications) from and sending messages to other nodes, and able to match subscriptions and publications in order to route messages to the right destinations. The broker can also temporarily store publications to satisfy future subscriptions.

CoAP pub/sub Client: A CoAP client that implements the CoAP pub/sub function set.

Topic: A unique identifier for a particular item being published and/or subscribed to. A broker uses the topics to match subscriptions to publications.

3. Architecture

3.1. CoAP pub/sub Architecture

Figure 1 shows the architecture of a CoAP pub/sub service. CoAP pub/sub Clients interact with a CoAP pub/sub Broker through the CoAP pub/sub interface which is hosted by the Broker. State information is updated between the Clients and the Broker. The CoAP pub/sub Broker performs a store-and-forward function of state updates between certain CoAP pub/sub Clients. Clients Subscribe to state updates which are Published by other Clients, and which are forwarded by the Broker to the subscribing clients. The CoAP pub/sub Broker also acts as a REST proxy, retaining the last state update provided by clients to supply in response to Read requests from Clients.

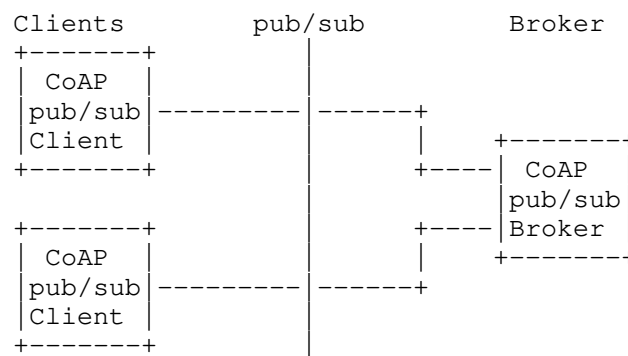


Figure 1: CoAP pub/sub Architecture

3.2. CoAP pub/sub Broker

A CoAP pub/sub Broker is a CoAP Server that exposes an interface for clients to use to initiate publish-subscribe interactions. Unlike clients, the broker needs to be reachable by all clients. The broker also needs to have sufficient resources (storage, bandwidth, etc.) to host CoAP resources, and potentially buffer messages, on behalf of the clients.

3.3. CoAP pub/sub Client

A CoAP pub/sub Client interacts with a CoAP pub/sub Broker using the CoAP pub/sub interface. Clients initiate all interactions with the CoAP pub/sub broker. A data source (e.g., sensor clients) can publish state updates to the broker and data sinks (e.g., actuator clients) can read from or subscribe to state updates from the broker. Application clients can make use of both publish and subscribe in order to exchange state updates with data sources and sinks.

3.4. CoAP pub/sub Topic

The clients and broker use topics to identify a particular resource or object in a publish-subscribe system. Topics are conventionally formed as a hierarchy, e.g. `"/sensors/weather/barometer/pressure"` or `"EP-33543/sen/3303/0/5700"`. The topics are hosted at the broker and all the clients using the broker share the same namespace for topics. A CoAP pub/sub topic has a reference path using URI path [RFC3986] construction, link attributes [RFC6690], and a representation of a value with specified content-formats. A CoAP pub/sub topic value may alternatively be a collection of one or more sub-topics, consisting of links to the sub-topic URIs and indicated by a link-format content-format.

4. CoAP pub/sub Function Set

This section defines the interfaces between a CoAP pub/sub Broker and pub/sub Clients, which is called the CoAP pub/sub Function Set. The examples throughout this section assume the use of CoAP [RFC7252]. A CoAP pub/sub Broker implementing this specification MUST support the DISCOVER, CREATE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, READ, and REMOVE operations defined in this section.

4.1. DISCOVER

CoAP pub/sub Clients discover CoAP pub/sub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [I-D.ietf-core-resource-directory]. A CoAP pub/sub Broker SHOULD indicate its presence and availability on a network by exposing a link to its pub/sub function set at its `.well-known/core` location [RFC6690]. A CoAP pub/sub broker MAY register its pub/sub function set location with a Resource Directory. Figure 2 shows an example of a client discovering a local pub/sub Function Set using CoAP Simple Discovery. A broker wishing to advertise the CoAP pub/sub Function Set for Simple Discovery or through a Resource Directory MUST use the link relation `rt="core.ps"`. A broker MAY advertise its supported content formats and other attributes in the link to its pub/sub function set.

A CoAP pub/sub Broker MAY offer the Discover interface to enable Clients to find topics of interest, either by topic name or by link attributes which may be registered when the topic is created. Figure 3 shows an example of a client looking for a topic with a resource type (rt) of "temperature" in the pub/sub function set /ps using the Discover interface. The client then receives the URI of the resource and its content-format.

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: /.well-known/core

URI Template: /{+ps/}{topic}/{/topic*}{?q*}

URI Template Variables:

/.well-known/core := for discovering the pub/sub function set (optional)

ps := pub/sub Function Set path (optional). The path of the pub/sub Function Set, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: application/link-format

The following response codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the broker resource. A pub/sub broker SHOULD use the value "/ps/" for the function set URI wherever possible.

Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

Client	Broker
<pre> ----- GET /.well-known/core?rt=core.ps -----> ---Content-Format: application/link-format--- <--2.05 Content "</ps/>;rt="core.ps";ct=40----</pre>	

Figure 2: Example of DISCOVER pub/sub function

Client	Broker
<pre> ----- GET /ps/?rt="temperature" -----> Content-Format: application/link-format <---2.05 Content </ps/currentTemp>;rt="temperature";ct=50 ---</pre>	

Figure 3: Example of DISCOVER topic

4.2. CREATE

Clients create topics on the broker using the CREATE interface. A client wishing to create a topic MUST use CoAP POST to the pub/sub function set location with a payload indicating the desired topic. The topic specification sent in the payload MUST use a supported serialization of the CoRE link format [RFC6690]. The target of the link MUST be a URI formatted string. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Format) link attribute in the link-format payload. The client MAY indicate the lifetime of the topic by including the Max-Age option in the CREATE request. Broker MUST return a response code of "2.01 Created" if the topic is created and return the created relative URI path via Location-Path options. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if a new topic can not be created. Broker SHOULD remove topics if the Max-Age of the topic is exceeded without any publishes to the topic. Broker SHOULD retain a topic indefinitely if the Max-Age option is elided or is set to zero upon topic creation. The lifetime of a topic MUST be refreshed upon create operations with a target of an existing topic.

Topics may be created as sub-topics of other topics. A client MAY create a topic with a ct (Content Format) link attribute value which

describes a supported serialization of the CoRE link format [RFC6690] such as application/link-format (ct=40) or its JSON or CBOR serializations. If a topic is created which describes a link serialization, that topic may then have sub-topics created under it as shown in Figure 5.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

URI Template: /{+ps/}{topic}/{/topic*}

URI Template Variables:

ps := pub/sub Function Set path (mandatory). The path of the pub/sub Function Set, as obtained from discovery. A pub/sub broker SHOULD use the value "ps" for this variable whenever possible.

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for this interface:

Success: 2.01 "Created". Successful Creation of the topic

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.03 "Forbidden". Topic already exists.

Failure: 4.06 "Not Acceptable". Unsupported content format for topic.

Figure 4 shows an example of a topic called "topic1" being successfully created.

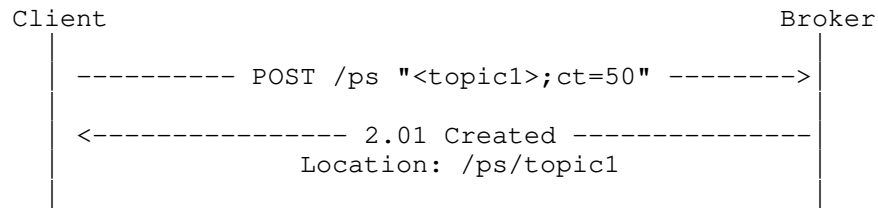


Figure 4: Example of CREATE topic

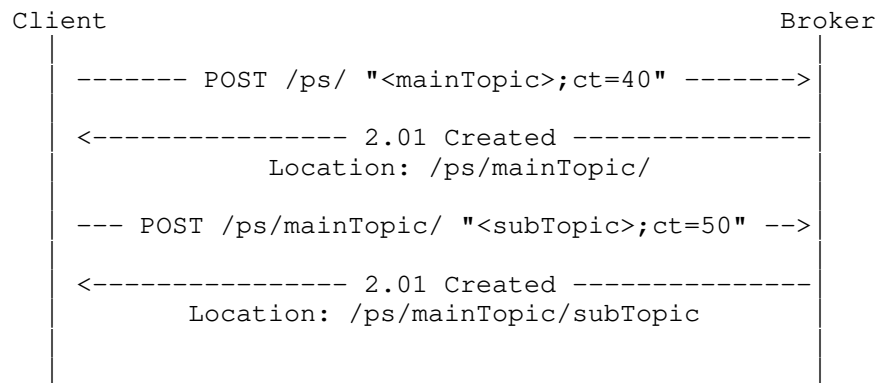


Figure 5: Example of CREATE sub-topic

4.3. PUBLISH

A CoAP pub/sub Client uses the PUBLISH interface for updating topics on the broker. The client **MUST** use the PUT method to publish state updates to the CoAP pub/sub Broker. A client **MUST** use the content format specified upon creation of a given topic to publish updates to that topic. The broker **MUST** reject publish operations which do not use the specified content format. A CoAP client publishing on a topic **MAY** indicate the maximum lifetime of the value by including the Max-Age option in the publish request. The broker **MUST** return a response code of "2.04 Changed" if the publish is accepted or "4.04 Not Found" if the topic does not exist. A broker **MAY** return "4.29 Too Many Requests" if simple flow control as described in Section 7 is implemented.

The Broker **MUST** notify all clients subscribed on a particular topic each time it receives a publish on that topic. An example is shown in Figure 7. If a client publishes to a broker with the Max-Age option, the broker **MUST** include the same value for the Max-Age option

in all notifications. A broker MUST use CoAP Notification as described in [RFC7641] to notify subscribed clients.

The PUBLISH interface is specified as follows:

Interaction: Client -> Broker

Method: PUT

URI Template: `/{"ps/"}{topic}/{topic*}`

URI Template Variables:

`ps` := pub/sub Function Set path (mandatory). The path of the pub/sub Function Set, as obtained from discovery.

`topic` := The desired topic to publish on.

Content-Format: Any valid CoAP content format

Payload: Representation of the topic value (CoAP resource state representation) in the indicated content format

The following response codes are defined for this interface:

Success: 2.04 "Changed". Successful publish, topic is updated

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 7).

Figure 6 shows an example of a new value being successfully published to the topic "topic1". See Figure 7 for an example of a broker forwarding a message from a publishing client to a subscribed client.

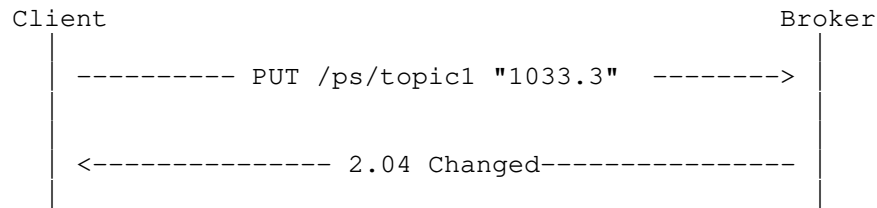


Figure 6: Example of PUBLISH

4.4. SUBSCRIBE

CoAP pub/sub Clients subscribe to topics on the Broker using CoAP Observe as described in [RFC7641]. A CoAP pub/sub Client wishing to Subscribe to a topic on a broker MUST use a CoAP GET with Observe registration. The Broker MAY add the client to a list of observers. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. The broker MUST accept Subscribe requests on a topic if the content format of the request matches the content format the topic was created with. The broker MAY accept Subscribe requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if it can not return the requested content format. If a Broker is unable to accept a new Subscription on a topic, it SHOULD return the appropriate response code without the Observe option as per as per [RFC7641] Section 4.1. There is no explicit maximum lifetime of a Subscription, thus a Broker may remove subscribers at any time. The Broker, upon removing a Subscriber, will transmit the appropriate response code without the Observe option, as per [RFC7641] Section 4.2, to the removed Subscriber.

The SUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

URI Template: `/{"ps/"}{topic}/{topic*}`

URI Template Variables:

`ps` := pub/sub Function Set path (mandatory). The path of the pub/sub Function Set, as obtained from discovery.

`topic` := The desired topic to subscribe to.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful subscribe, current value included

Success: 2.04 "No Content". Successful subscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content format.

Figure 7 shows an example of Client2 subscribing to "topic1" and receiving a response from the broker, with a subsequent notification. The subscribe response from the broker uses the last stored value associated with the topic1. The notification from the broker is sent in response to the publish received from Client1.

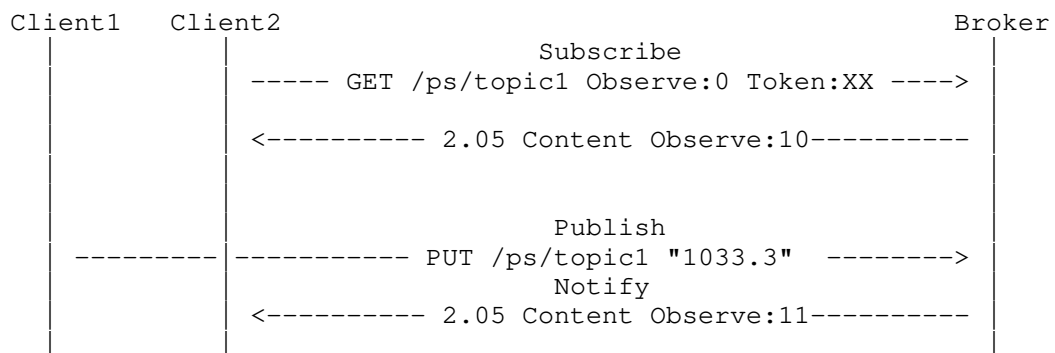


Figure 7: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

CoAP pub/sub Clients unsubscribe from topics on the Broker using the CoAP Cancel Observation operation. A CoAP pub/sub Client wishing to unsubscribe to a topic on a Broker MUST either use CoAP GET with Observe using an Observe parameter of 1 or send a CoAP Reset message in response to a publish, as per [RFC7641].

The UNSUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

URI Template: /{+ps/}{topic}/{/topic*}

URI Template Variables:

ps := pub/sub Function Set path (mandatory). The path of the pub/sub Function Set, as obtained from discovery.

topic := The desired topic to unsubscribe from.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful unsubscribe, current value included

Success: 2.04 "No Content". Successful unsubscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 8 shows an example of a client unsubscribe using the Observe=1 cancellation method.

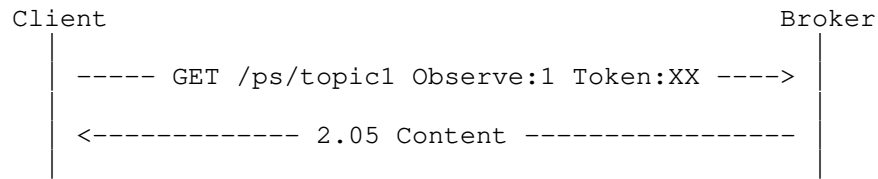


Figure 8: Example of UNSUBSCRIBE

4.6. READ

A CoAP pub/sub client wishing to obtain only the most recent published value on a topic MAY use the READ interface. For reading, the client uses the CoAP GET method. The broker MUST accept Read requests on a topic if the content format of the request matches the content format the topic was created with. The broker MAY accept Read requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the topic to be valid since the last publish. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if the broker can not return the requested content format.

The READ interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: `/ps/{topic}/topic*`

URI Template Variables:

`ps` := pub/sub Function Set path (mandatory). The path of the pub/sub Function Set, as obtained from discovery.

`topic` := The desired topic to READ.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful READ, current value included

Success: 2.04 "No Content". Topic exists, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content-format.

Figure 9 shows an example of a successful READ from topic1, followed by a Publish on the topic, followed at some time later by a read of the updated value from the recent Publish.

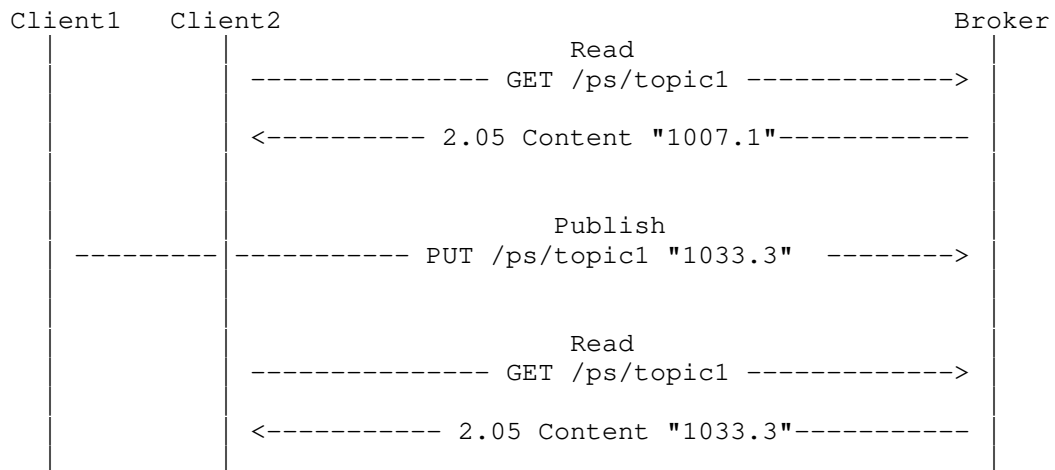


Figure 9: Example of READ

4.7. REMOVE

A CoAP pub/sub Client wishing to remove a topic MAY use the CoAP Delete operation on the URI of the topic. The CoAP pub/sub Broker MUST return "2.02 Deleted" if the remove operation is successful. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if the topic can not be removed. When a topic is removed for any reason, the Broker SHOULD return the response code 4.04 Not Found and remove all of the observers from the list of observers as per as per [RFC7641] Section 3.2.

The REMOVE interface is specified as follows:

Interaction: Client -> Broker

Method: DELETE

URI Template: `/{"+ps/"}{topic}/{/topic*}`

URI Template Variables:

`ps` := pub/sub Function Set path (mandatory). The path of the pub/sub Function Set, as obtained from discovery.

`topic` := The desired topic to REMOVE.

Content-Format: None

Response Payload: None

The following response codes are defined for this interface:

Success: 2.02 "Deleted". Successful remove

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 10 shows a successful remove of topic1.

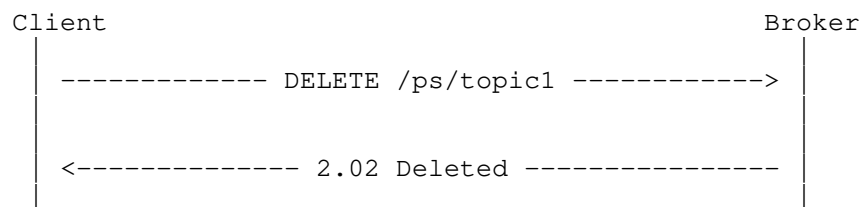


Figure 10: Example of REMOVE

5. CoAP pub/sub Operation with Resource Directory

A CoAP pub/sub Broker may register a pub/sub Function Set with a Resource Directory. A pub/sub Client may use an RD to discover a pub/sub Broker.

A CoAP pub/sub Client may register CoRE Links [RFC6690] to created pub/sub Topics with an RD. A pub/sub Client may use an RD to discover pub/sub Topics. A client which registers pub/sub Topics with an RD MUST use the context relation (con) [I-D.ietf-core-resource-directory] to indicate that the context of the registered links is the pub/sub Broker.

6. Sleep-Wake Operation

CoAP pub/sub provides a way for client nodes to sleep between operations, conserving energy during idle periods. This is made possible by shifting the server role to the broker, allowing the broker to be always-on and respond to requests from other clients while a particular client is sleeping.

For example, the broker will retain the last state update received from a sleeping client, in order to supply the most recent state update to other clients in response to read and subscribe operations.

Likewise, the broker will retain the last state update received on the topic such that a sleeping client, upon waking, can perform a read operation to the broker to update its own state from the most recent system state update.

7. Simple Flow Control

Since the broker node has to potentially send a large amount of notification messages for each publish message and it may be serving a large amount of subscribers and publishers simultaneously, the broker may become overwhelmed if it receives many publish messages to popular topics in a short period of time.

If the broker is unable to serve a certain client that is sending publish messages too fast, the broker MUST respond with Response Code 4.29, "Too Many Requests". This Response Code is like HTTP 429 "Too Many Requests" but uses the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry. The broker MAY stop creating notifications from the publish messages from this client and to this topic for the indicated time.

If a client receives the 4.29 Response Code from the broker for a publish message to a topic, it MUST NOT send new publish messages to the broker on the same topic before the time indicated in Max-Age has passed.

8. Security Considerations

CoAP pub/sub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP pub/sub broker and the clients SHOULD authenticate each other and enforce access control policies. A malicious client could subscribe to data it is not authorized to or mount a denial of service attack against the broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP pub/sub broker introduces challenges for the use of end-to-end security between for example a client device on a sensor network and a client application running in a cloud-based server infrastructure since brokers terminate the exchange. While running separate DTLS sessions from the client device to the broker and from broker to client application protects confidentiality on those paths, the client device does not know whether the commands coming from the broker are actually coming from the client application. Similarly, a client application requesting data does not know whether the data originated on the client device. For scenarios where end-to-end security is desirable the use of application layer security is unavoidable. Application layer security would then provide a guarantee to the client device that any request originated at the client application. Similarly, integrity protected sensor data from a client device will also provide guarantee to the client application that the data originated on the client device itself. The protected data can also be verified by the intermediate broker ensuring that it stores/caches correct request/response and no malicious messages/requests are accepted. The broker would still be able to perform aggregation of data/requests collected.

Depending on the level of trust users and system designers place in the CoAP pub/sub broker, the use of end-to-end object security is RECOMMENDED [I-D.selander-ace-object-security].

When only end-to-end encryption is necessary and the CoAP Broker is trusted, Payload Only Protection (Mode:PAYL) could be used. The Publisher would wrap only the payload before sending it to the broker and set the option Content-Format to application/smpayl. Upon receipt, the Broker can read the unencrypted CoAP header to forward it to the subscribers.

9. IANA Considerations

This document registers one attribute value in the Resource Type (rt=) registry established with [RFC6690] and appends to the definition of one CoAP Response Code in the CoRE Parameters Registry.

9.1. Resource Type value 'core.ps'

- o Attribute Value: core.ps
- o Description: Section 4 of [[This document]]
- o Reference: [[This document]]
- o Notes: None

9.2. Response Code value '2.04'

- o Response Code: 2.04
- o Description: Add No Content response to GET to the existing definition of the 2.04 response code.
- o Reference: [[This document]]
- o Notes: None

9.3. Response Code value '4.29'

- o Response Code: 4.29
- o Description: This error code is used by a server to indicate that a client is making too many requests on a resource.
- o Reference: [[This document]]
- o Notes: None

10. Acknowledgements

The authors would like to thank Hannes Tschofenig, Zach Shelby, Mohit Sethi, Peter van der Stok, Tim Kellogg, Anders Eriksson, Goran Selander, Mikko Majanen, and Olaf Bergmann for their contributions and reviews.

11. References

11.1. Normative References

- [I-D.ietf-core-resource-directory]
Shelby, Z., Kostner, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-05 (work in progress), October 2015.
- [I-D.selander-ace-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-03 (work in progress), October 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<http://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

11.2. Informative References

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.

Authors' Addresses

Michael Koster
ARM Limited

Email: Michael.Koster@arm.com

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson

Email: jaime.jimenez@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2017

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson
July 7, 2016

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-koster-core-coap-pubsub-05

Abstract

The Constrained Application Protocol (CoAP), and related extensions are intended to support machine-to-machine communication in systems where one or more nodes are resource constrained, in particular for low power wireless sensor networks. This document defines a publish-subscribe broker for CoAP that extends the capabilities of CoAP for supporting nodes with long breaks in connectivity and/or up-time.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 8, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Architecture	4
3.1. CoAP pubsub Architecture	4
3.2. CoAP pubsub Broker	4
3.3. CoAP pubsub Client	5
3.4. CoAP pubsub Topic	5
3.5. Brokerless pubsub	5
4. CoAP pubsub Function Set	6
4.1. DISCOVER	6
4.2. CREATE	8
4.3. PUBLISH	10
4.4. SUBSCRIBE	12
4.5. UNSUBSCRIBE	14
4.6. READ	15
4.7. REMOVE	16
5. CoAP pubsub Operation with Resource Directory	17
6. Sleep-Wake Operation	18
7. Simple Flow Control	18
8. Security Considerations	19
9. IANA Considerations	20
9.1. Resource Type value 'core.ps'	20
9.2. Response Code value '2.04'	20
9.3. Response Code value '4.29'	20
10. Acknowledgements	21
11. References	21
11.1. Normative References	21
11.2. Informative References	22
Authors' Addresses	22

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports machine-to-machine communication across networks of constrained devices. CoAP uses a request/response model where clients make requests to servers in order to request actions on resources. Depending on the situation the same device may act either as a server or a client.

One important class of constrained devices includes devices that are intended to run for years from a small battery, or by scavenging energy from their environment. These devices have limited

reachability because they spend most of their time in a sleeping state with no network connectivity. Devices may also have limited reachability due to certain middle-boxes, such as Network Address Translators (NATs) or firewalls. Such middle-boxes often prevent connecting to a device from the Internet unless the connection was initiated by the device.

This document specifies the means for nodes with limited reachability to communicate using simple extensions to CoAP. The extensions enable publish-subscribe communication using a broker node that enables store-and-forward messaging between two or more nodes. Furthermore the extensions facilitate many-to-many communication using CoAP.

2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [RFC2119].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format [RFC6570] is used to describe the REST interfaces defined in this specification.

This specification makes use of the following additional terminology:

Publish-Subscribe (pubsub): A messaging paradigm where messages are published to a broker and potential receivers can subscribe to the broker to receive messages. The publishers do not (need to) know where the message will be eventually sent: the publications and subscriptions are matched by a broker and publications are delivered by the broker to subscribed receivers.

CoAP pubsub function set: A group of well-known REST resources that together provide the CoAP pubsub service.

CoAP pubsub Broker: A server node capable of receiving messages (publications) from and sending messages to other nodes, and able to match subscriptions and publications in order to route messages to the right destinations. The broker can also temporarily store publications to satisfy future subscriptions.

CoAP pubsub Client: A CoAP client that implements the CoAP pubsub function set.

Topic: A unique identifier for a particular item being published and/or subscribed to. A broker uses the topics to match subscriptions to publications.

3. Architecture

3.1. CoAP pubsub Architecture

Figure 1 shows the architecture of a CoAP pubsub service. CoAP pubsub Clients interact with a CoAP pubsub Broker through the CoAP pubsub interface which is hosted by the Broker. State information is updated between the Clients and the Broker. The CoAP pubsub Broker performs a store-and-forward function of state updates between certain CoAP pubsub Clients. Clients Subscribe to state updates which are Published by other Clients, and which are forwarded by the Broker to the subscribing clients. The CoAP pubsub Broker also acts as a REST proxy, retaining the last state update provided by clients to supply in response to Read requests from Clients.

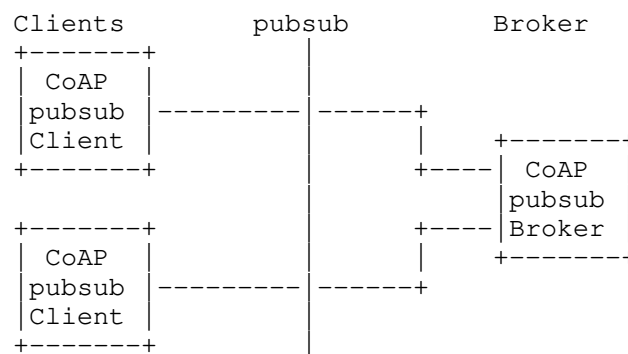


Figure 1: CoAP pubsub Architecture

3.2. CoAP pubsub Broker

A CoAP pubsub Broker is a CoAP Server that exposes an interface for clients to use to initiate publish-subscribe interactions. Unlike clients, the broker needs to be reachable by all clients. The broker also needs to have sufficient resources (storage, bandwidth, etc.) to host CoAP resources, and potentially buffer messages, on behalf of the clients.

3.3. CoAP pubsub Client

A CoAP pubsub Client interacts with a CoAP pubsub Broker using the CoAP pubsub interface. Clients initiate all interactions with the CoAP pubsub broker. A data source (e.g., sensor clients) can publish state updates to the broker and data sinks (e.g., actuator clients) can read from or subscribe to state updates from the broker. Application clients can make use of both publish and subscribe in order to exchange state updates with data sources and sinks.

3.4. CoAP pubsub Topic

The clients and broker use topics to identify a particular resource or object in a publish-subscribe system. Topics are conventionally formed as a hierarchy, e.g. `"/sensors/weather/barometer/pressure"` or `"EP-33543/sen/3303/0/5700"`. The topics are hosted at the broker and all the clients using the broker share the same namespace for topics. A CoAP pubsub topic has a reference path using URI path [RFC3986] construction, link attributes [RFC6690], and a representation of a value with specified content-formats. A CoAP pubsub topic value may alternatively be a collection of one or more sub-topics, consisting of links to the sub-topic URIs and indicated by a link-format content-format.

3.5. Brokerless pubsub

Figure 2 shows an arrangement for using CoAP pubsub in a "brokerless" configuration between peer nodes. Nodes in a brokerless system act as both broker and client. The Broker interface in a brokerless node may be pre-configured with topics that expose services and resources. Brokerless peer nodes can be mixed with client and broker nodes in a system with full interoperability.

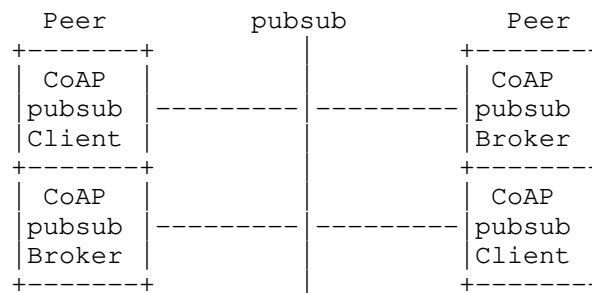


Figure 2: Brokerless pubsub

4. CoAP pubsub Function Set

This section defines the interfaces between a CoAP pubsub Broker and pubsub Clients, which is called the CoAP pubsub Function Set. The examples throughout this section assume the use of CoAP [RFC7252]. A CoAP pubsub Broker implementing this specification MUST support the DISCOVER, CREATE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, READ, and REMOVE operations defined in this section.

4.1. DISCOVER

CoAP pubsub Clients discover CoAP pubsub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [I-D.ietf-core-resource-directory]. A CoAP pubsub Broker SHOULD indicate its presence and availability on a network by exposing a link to its pubsub function set at its .well-known/core location [RFC6690]. A CoAP pubsub broker MAY register its pubsub function set location with a Resource Directory. Figure 3 shows an example of a client discovering a local pubsub Function Set using CoAP Simple Discovery. A broker wishing to advertise the CoAP pubsub Function Set for Simple Discovery or through a Resource Directory MUST use the link relation rt="core.ps". A broker MAY advertise it's supported content formats and other attributes in the link to it's pubsub function set.

A CoAP pubsub Broker MAY offer the Discover interface to enable Clients to find topics of interest, either by topic name or by link attributes which may be registered when the topic is created. Figure 4 shows an example of a client looking for a topic with a resource type (rt) of "temperature" in the pubsub function set /ps using the Discover interface. The client then receives the URI of the resource and its content-format.

A CoAP pubsub Broker MAY expose the Discover interface through the .well-known/core resource. Links to topics may be exposed at .well-known/core in addition to links to the pubsub function set. Figure 5 shows an example of topic discovery through .well-known/core.

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: /.well-known/core

URI Template: /{+ps/}{topic}/{/topic*}{?q*}

URI Template Variables:

`/.well-known/core` := for discovering the pubsub function set (optional)

`ps` := pubsub Function Set path (optional). The path of the pubsub Function Set, as obtained from discovery, used to discover topics.

`topic` := The desired topic to return links for (optional).

`q` := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: application/link-format

The following response codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the broker resource. A pubsub broker SHOULD use the value `"/ps/"` for the function set URI wherever possible.

Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

Client	Broker
<pre> ----- GET /.well-known/core?rt=core.ps -----> ---Content-Format: application/link-format--- <--2.05 Content "</ps/>;rt="core.ps";ct=40----</pre>	<pre> -----> ---Content-Format: application/link-format--- <--2.05 Content "</ps/>;rt="core.ps";ct=40----</pre>

Figure 3: Example of DISCOVER pubsub function

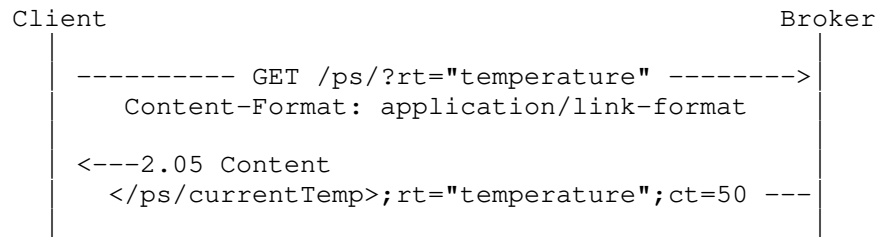


Figure 4: Example of DISCOVER topic

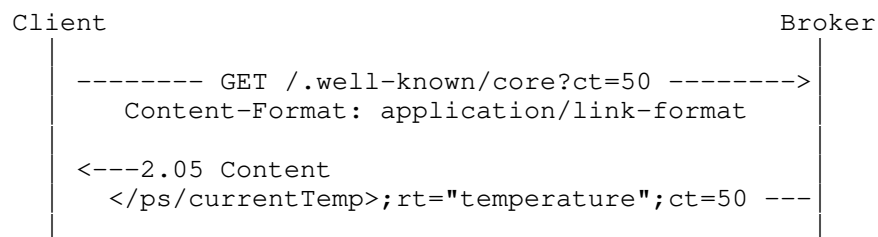


Figure 5: Example of DISCOVER topic

4.2. CREATE

Clients create topics on the broker using the CREATE interface. A client wishing to create a topic MUST use CoAP POST to the pubsub function set location with a payload indicating the desired topic. The topic specification sent in the payload MUST use a supported serialization of the CoRE link format [RFC6690]. The target of the link MUST be a URI formatted string. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Format) link attribute in the link-format payload. The client MAY indicate the lifetime of the topic by including the Max-Age option in the CREATE request. Broker MUST return a response code of "2.01 Created" if the topic is created and return the created relative URI path via Location-Path options. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if a new topic can not be created. Broker SHOULD remove topics if the Max-Age of the topic is exceeded without any publishes to the topic. Broker SHOULD retain a topic indefinitely if the Max-Age option is elided or is set to zero upon topic creation. The lifetime of a topic MUST be refreshed upon create operations with a target of an existing topic.

Topics may be created as sub-topics of other topics. A client MAY create a topic with a ct (Content Format) link attribute value which describes a supported serialization of the CoRE link format [RFC6690] such as application/link-format (ct=40) or its JSON or CBOR serializations. If a topic is created which describes a link serialization, that topic may then have sub-topics created under it as shown in Figure 7.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

URI Template: /{+ps/}{topic}/{/topic*}

URI Template Variables:

ps := pubsub Function Set path (mandatory). The path of the pubsub Function Set, as obtained from discovery. A pubsub broker SHOULD use the value "ps" for this variable whenever possible.

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for this interface:

Success: 2.01 "Created". Successful Creation of the topic

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.03 "Forbidden". Topic already exists.

Failure: 4.06 "Not Acceptable". Unsupported content format for topic.

Figure 6 shows an example of a topic called "topic1" being successfully created.

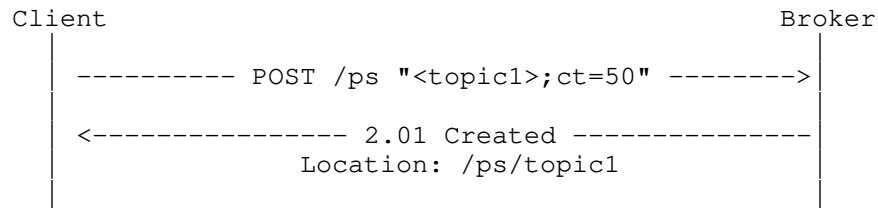


Figure 6: Example of CREATE topic

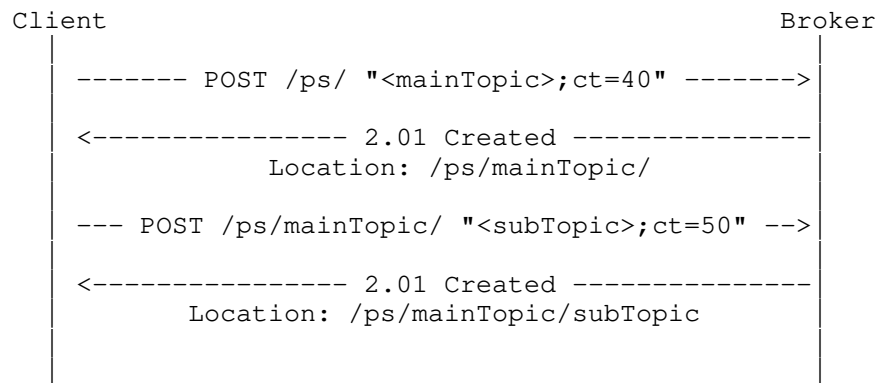


Figure 7: Example of CREATE sub-topic

4.3. PUBLISH

A CoAP pubsub Client uses the PUBLISH interface for updating topics on the broker. The client MUST use the PUT method to publish state updates to the CoAP pubsub Broker. A client MUST use the content format specified upon creation of a given topic to publish updates to that topic. The broker MUST reject publish operations which do not use the specified content format. A CoAP client publishing on a topic MAY indicate the maximum lifetime of the value by including the Max-Age option in the publish request. The broker MUST return a response code of "2.04 Changed" if the publish is accepted or "4.04 Not Found" if the topic does not exist. A broker MAY return "4.29 Too Many Requests" if simple flow control as described in Section 7 is implemented.

The Broker MUST notify all clients subscribed on a particular topic each time it receives a publish on that topic. An example is shown in Figure 9. If a client publishes to a broker with the Max-Age option, the broker MUST include the same value for the Max-Age option

in all notifications. A broker MUST use CoAP Notification as described in [RFC7641] to notify subscribed clients.

The PUBLISH interface is specified as follows:

Interaction: Client -> Broker

Method: PUT

URI Template: `/{"ps/"}{topic}/{topic*}`

URI Template Variables:

`ps` := pubsub Function Set path (mandatory). The path of the pubsub Function Set, as obtained from discovery.

`topic` := The desired topic to publish on.

Content-Format: Any valid CoAP content format

Payload: Representation of the topic value (CoAP resource state representation) in the indicated content format

The following response codes are defined for this interface:

Success: 2.04 "Changed". Successful publish, topic is updated

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 7).

Figure 8 shows an example of a new value being successfully published to the topic "topic1". See Figure 9 for an example of a broker forwarding a message from a publishing client to a subscribed client.

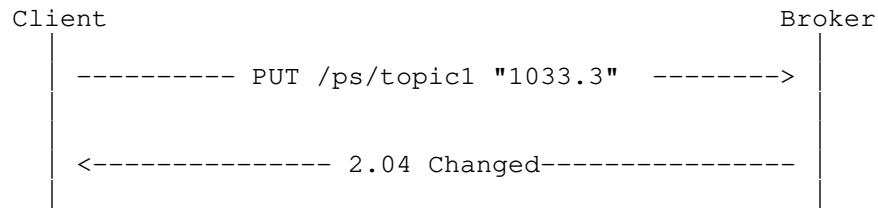


Figure 8: Example of PUBLISH

4.4. SUBSCRIBE

CoAP pubsub Clients subscribe to topics on the Broker using CoAP Observe as described in [RFC7641]. A CoAP pubsub Client wishing to Subscribe to a topic on a broker MUST use a CoAP GET with Observe registration. The Broker MAY add the client to a list of observers. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. The broker MUST accept Subscribe requests on a topic if the content format of the request matches the content format the topic was created with. The broker MAY accept Subscribe requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if it can not return the requested content format. If a Broker is unable to accept a new Subscription on a topic, it SHOULD return the appropriate response code without the Observe option as per as per [RFC7641] Section 4.1. There is no explicit maximum lifetime of a Subscription, thus a Broker may remove subscribers at any time. The Broker, upon removing a Subscriber, will transmit the appropriate response code without the Observe option, as per [RFC7641] Section 4.2, to the removed Subscriber.

The SUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

URI Template: `/{"ps/"}{topic}/{topic*}`

URI Template Variables:

`ps` := pubsub Function Set path (mandatory). The path of the pubsub Function Set, as obtained from discovery.

`topic` := The desired topic to subscribe to.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful subscribe, current value included

Success: 2.04 "No Content". Successful subscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content format.

Figure 9 shows an example of Client2 subscribing to "topic1" and receiving a response from the broker, with a subsequent notification. The subscribe response from the broker uses the last stored value associated with the topic1. The notification from the broker is sent in response to the publish received from Client1.

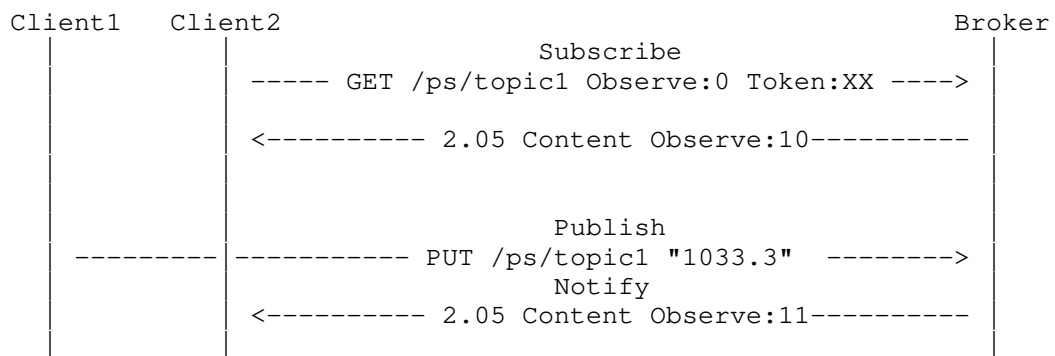


Figure 9: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

CoAP pubsub Clients unsubscribe from topics on the Broker using the CoAP Cancel Observation operation. A CoAP pubsub Client wishing to unsubscribe to a topic on a Broker MUST either use CoAP GET with Observe using an Observe parameter of 1 or send a CoAP Reset message in response to a publish, as per [RFC7641].

The UNSUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

URI Template: /{+ps/}{topic}/{/topic*}

URI Template Variables:

ps := pubsub Function Set path (mandatory). The path of the pubsub Function Set, as obtained from discovery.

topic := The desired topic to unsubscribe from.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful unsubscribe, current value included

Success: 2.04 "No Content". Successful unsubscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 10 shows an example of a client unsubscribe using the Observe=1 cancellation method.

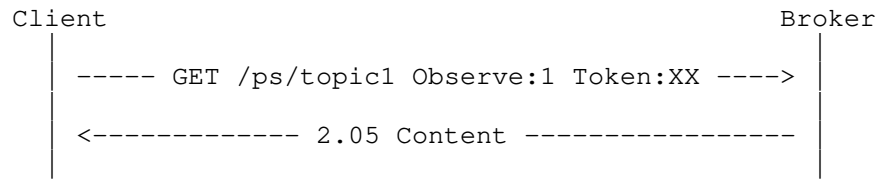


Figure 10: Example of UNSUBSCRIBE

4.6. READ

A CoAP pubsub client wishing to obtain only the most recent published value on a topic MAY use the READ interface. For reading, the client uses the CoAP GET method. The broker MUST accept Read requests on a topic if the content format of the request matches the content format the topic was created with. The broker MAY accept Read requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the topic to be valid since the last publish. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if the broker can not return the requested content format.

The READ interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: `/[+ps/]{topic}/{/topic*}`

URI Template Variables:

`ps` := pubsub Function Set path (mandatory). The path of the pubsub Function Set, as obtained from discovery.

`topic` := The desired topic to READ.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful READ, current value included

Success: 2.04 "No Content". Topic exists, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content-format.

Figure 11 shows an example of a successful READ from topic1, followed by a Publish on the topic, followed at some time later by a read of the updated value from the recent Publish.

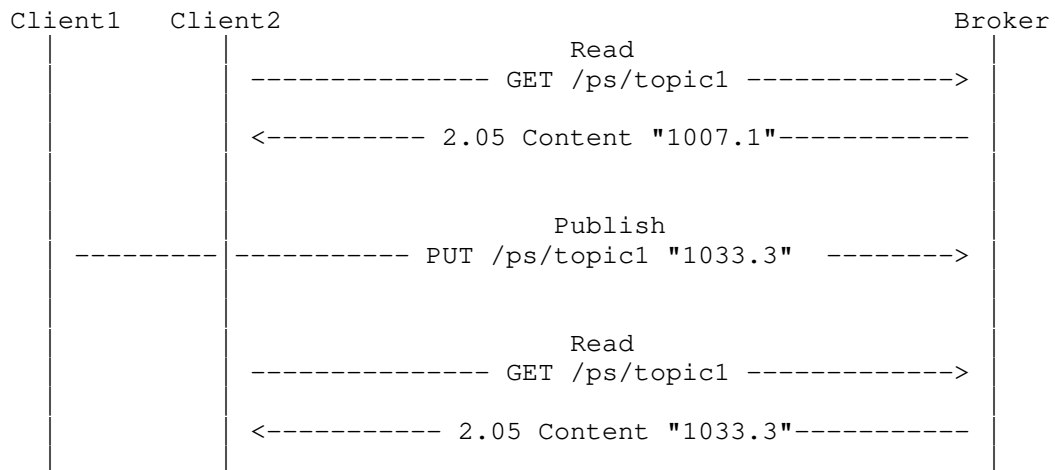


Figure 11: Example of READ

4.7. REMOVE

A CoAP pubsub Client wishing to remove a topic MAY use the CoAP Delete operation on the URI of the topic. The CoAP pubsub Broker MUST return "2.02 Deleted" if the remove operation is successful. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if the topic can not be removed. When a topic is removed for any reason, the Broker SHOULD return the response code 4.04 Not Found and remove all of the observers from the list of observers as per as per [RFC7641] Section 3.2.

The REMOVE interface is specified as follows:

Interaction: Client -> Broker

Method: DELETE

URI Template: `/{"ps/"}{topic}/{topic*}`

URI Template Variables:

`ps` := pubsub Function Set path (mandatory). The path of the pubsub Function Set, as obtained from discovery.

`topic` := The desired topic to REMOVE.

Content-Format: None

Response Payload: None

The following response codes are defined for this interface:

Success: 2.02 "Deleted". Successful remove

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 12 shows a successful remove of topic1.

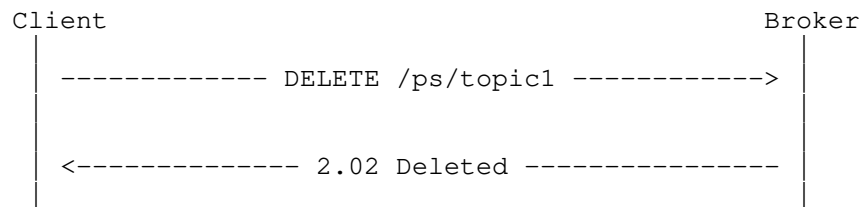


Figure 12: Example of REMOVE

5. CoAP pubsub Operation with Resource Directory

A CoAP pubsub Broker may register a pubsub Function Set with a Resource Directory. A pubsub Client may use an RD to discover a pubsub Broker.

A CoAP pubsub Client may register links [RFC6690] with a Resource Directory to enable discovery of created pubsub topics. A pubsub Client may use an RD to discover pubsub Topics. A client which registers pubsub Topics with an RD MUST use the context relation (con) [I-D.ietf-core-resource-directory] to indicate that the context of the registered links is the pubsub Broker.

A CoAP pubsub Broker may alternatively register links to its topics to a Resource Directory by triggering the RD to retrieve it's links from .well-known/core. In order to use this method, the links must first be exposed in the .well-known/core of the pubsub broker. See Section 4.1 in this document.

The pubsub broker triggers the RD to retrieve its links by sending a POST with an empty payload to the .well-known/core of the Resource Directory. The RD server will then retrieve the links from the .well-known/core of the pubsub broker and incorporate them into the Resource Directory. See [I-D.ietf-core-resource-directory] for further details.

6. Sleep-Wake Operation

CoAP pubsub provides a way for client nodes to sleep between operations, conserving energy during idle periods. This is made possible by shifting the server role to the broker, allowing the broker to be always-on and respond to requests from other clients while a particular client is sleeping.

For example, the broker will retain the last state update received from a sleeping client, in order to supply the most recent state update to other clients in response to read and subscribe operations.

Likewise, the broker will retain the last state update received on the topic such that a sleeping client, upon waking, can perform a read operation to the broker to update its own state from the most recent system state update.

7. Simple Flow Control

Since the broker node has to potentially send a large amount of notification messages for each publish message and it may be serving a large amount of subscribers and publishers simultaneously, the broker may become overwhelmed if it receives many publish messages to popular topics in a short period of time.

If the broker is unable to serve a certain client that is sending publish messages too fast, the broker MUST respond with Response Code 4.29, "Too Many Requests". This Response Code is like HTTP 429 "Too

Many Requests" but uses the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry. The broker MAY stop creating notifications from the publish messages from this client and to this topic for the indicated time.

If a client receives the 4.29 Response Code from the broker for a publish message to a topic, it MUST NOT send new publish messages to the broker on the same topic before the time indicated in Max-Age has passed.

8. Security Considerations

CoAP pubsub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP pubsub broker and the clients SHOULD authenticate each other and enforce access control policies. A malicious client could subscribe to data it is not authorized to or mount a denial of service attack against the broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP pubsub broker introduces challenges for the use of end-to-end security between for example a client device on a sensor network and a client application running in a cloud-based server infrastructure since brokers terminate the exchange. While running separate DTLS sessions from the client device to the broker and from broker to client application protects confidentiality on those paths, the client device does not know whether the commands coming from the broker are actually coming from the client application. Similarly, a client application requesting data does not know whether the data originated on the client device. For scenarios where end-to-end security is desirable the use of application layer security is unavoidable. Application layer security would then provide a guarantee to the client device that any request originated at the client application. Similarly, integrity protected sensor data from a client device will also provide guarantee to the client application that the data originated on the client device itself. The protected data can also be verified by the intermediate broker ensuring that it stores/caches correct request/response and no malicious messages/requests are accepted. The broker would still be able to perform aggregation of data/requests collected.

Depending on the level of trust users and system designers place in the CoAP pubsub broker, the use of end-to-end object security is RECOMMENDED [I-D.selander-ace-object-security].

When only end-to-end encryption is necessary and the CoAP Broker is trusted, Payload Only Protection (Mode:PAYL) could be used. The Publisher would wrap only the payload before sending it to the broker and set the option Content-Format to application/smpayl. Upon receipt, the Broker can read the unencrypted CoAP header to forward it to the subscribers.

9. IANA Considerations

This document registers one attribute value in the Resource Type (rt=) registry established with [RFC6690] and appends to the definition of one CoAP Response Code in the CoRE Parameters Registry.

9.1. Resource Type value 'core.ps'

- o Attribute Value: core.ps
- o Description: Section 4 of [[This document]]
- o Reference: [[This document]]
- o Notes: None

9.2. Response Code value '2.04'

- o Response Code: 2.04
- o Description: Add No Content response to GET to the existing definition of the 2.04 response code.
- o Reference: [[This document]]
- o Notes: None

9.3. Response Code value '4.29'

- o Response Code: 4.29
- o Description: This error code is used by a server to indicate that a client is making too many requests on a resource.
- o Reference: [[This document]]
- o Notes: None

10. Acknowledgements

The authors would like to thank Hannes Tschofenig, Zach Shelby, Mohit Sethi, Peter van der Stok, Tim Kellogg, Anders Eriksson, Goran Selander, Mikko Majanen, and Olaf Bergmann for their contributions and reviews.

11. References

11.1. Normative References

- [I-D.ietf-core-resource-directory]
Shelby, Z., Kostner, M., Bormann, D., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-07 (work in progress), March 2016.
- [I-D.selander-ace-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-05 (work in progress), July 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<http://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

11.2. Informative References

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.

Authors' Addresses

Michael Koster
SmartThings

Email: Michael.Koster@smarththings.com

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson

Email: jaime.jimenez@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

J. Mattsson
J. Fornehed
G. Selander
F. Palombini
Ericsson
March 21, 2016

Controlling Actuators with CoAP
draft-mattsson-core-coap-actuators-01

Abstract

Being able to trust information from sensors and to securely control actuators is essential in a world of connected and networking things interacting with the physical world. In this memo we show that just using COAP with a security protocol like DTLS, TLS, or OSCOAP is not enough. We describe several serious attacks any on-path attacker can do, and discusses tougher requirements and mechanisms to mitigate the attacks. While this document is focused on actuators, one of the attacks applies equally well to sensors using DTLS.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Attacks	3
2.1. The Block Attack	3
2.2. The Request Delay Attack	4
2.3. The Response Delay and Mismatch Attack	7
2.4. The Relay Attack	10
3. The Repeat Option	11
4. IANA Considerations	13
5. Security Considerations	13
6. Acknowledgements	13
7. References	14
7.1. Normative References	14
7.2. Informative References	14
Authors' Addresses	14

1. Introduction

Being able to trust information from sensors and to securely control actuators is essential in a world of connected and networking things interacting with the physical world. One protocol used to interact with sensors and actuators is the Constrained Application Protocol (CoAP) [RFC7252]. Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [RFC6347], TLS [RFC5246], or OSCOAP [I-D.selander-ace-object-security] to protect CoAP, where the choice of security protocol depends on the transport protocol and the presence of intermediaries. The use of CoAP over UDP and DTLS is specified in [RFC6347] and the use of CoAP over TCP and TLS is specified in [I-D.ietf-core-coap-tcp-tls]. OSCOAP protects CoAP end-to-end with the use of COSE [I-D.ietf-cose-msg] and the CoAP Object-Security option [I-D.selander-ace-object-security], and can therefore be used over any transport. In this document we show that protecting CoAP with a security protocol is not enough to securely control actuators. We describe several serious attacks any on-path attacker (i.e. not only "trusted" intermediaries) can do, and discusses tougher requirements and mechanisms to mitigate the attacks. The request delay attack (valid for DTLS, TLS, and OSCOAP and described in Section 2.2) lets an attacker control an actuator at a much later time than the client anticipated. The response delay and mismatch attack (valid for DTLS and described in Section 2.3) lets an attacker

respond to a client with a response meant for an older request. In Section 3, a new CoAP Option, the Repeat Option, mitigating the delay attack is specified.

2. Attacks

Internet-of-Things (IoT) deployments valuing security and privacy, MUST use a security protocol such as DTLS, TLS, or OSCOAP to protect CoAP. This is especially true for deployments of actuators where attacks often (but not always) have serious consequences. The attacks described in this section are made under the assumption that CoAP is already protected with a security protocol such as DTLS, TLS, or OSCOAP, as an attacker otherwise can easily forge false requests and responses.

2.1. The Block Attack

An on-path attacker can block the delivery of any number of requests or responses. The attack can also be performed by an attacker jamming the lower layer radio protocol. This is true even if a security protocol like DTLS, TLS, or OSCOAP is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS and TLS, proxies have access to the complete CoAP message, and with OSCOAP, the CoAP header and several CoAP options are not encrypted. In both security protocols, the IP-addresses, ports, and CoAP message lengths are available to all on-path attackers, which may be enough to determine the server, resource, and command. The block attack is illustrated in Figure 1 and 2.

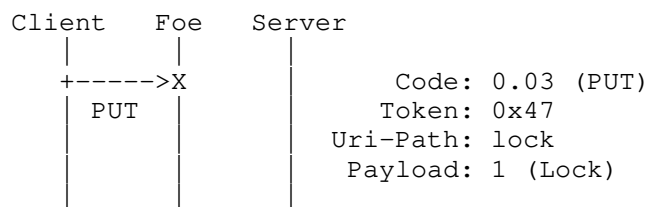


Figure 1: Blocking a Request

Where 'X' means the attacker is blocking delivery of the message.

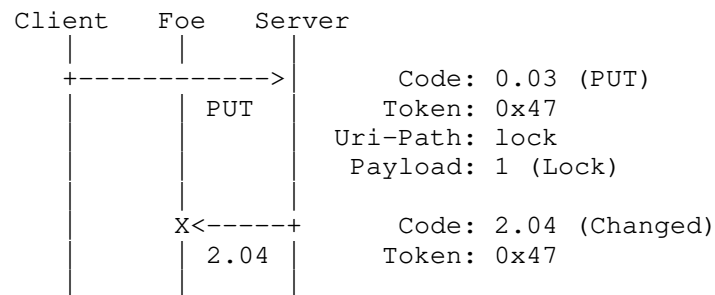


Figure 2: Blocking a Response

While blocking requests to, or responses from, a sensor is just a denial of service attack, blocking a request to, or a response from, an actuator results in the client losing information about the server's status. If the actuator e.g. is a lock (door, car, etc.), the attack results in the client not knowing (except by using out-of-band information) whether the lock is unlocked or locked, just like the observer in the famous Schrodinger's cat thought experiment. Due to the nature of the attack, the client cannot distinguish the attack from connectivity problems, offline servers, or unexpected behavior from middle boxes such as NATs and firewalls.

Remedy: Any IoT deployment of actuators where confirmation is important **MUST** notify the user upon reception of the response, or warn the user when a response is not received.

2.2. The Request Delay Attack

An on-path attacker may not only block packets, but can also delay the delivery of any packet (request or response) by a chosen amount of time. If CoAP is used over a reliable and ordered transport such as TCP with TLS or OSCOAP, no messages can be delivered before the delayed message. If CoAP is used over an unreliable and unordered transport such as UDP with DTLS, or OSCOAP, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. When CoAP is used over UDP, both DTLS and OSCOAP allow out-of-order delivery and uses sequence numbers together with a replay window to protect against replay attacks. The replay window has a default length of 64 in both DTLS and OSCOAP. The attacker can control the replay window by blocking some or all other packets. By first delaying a request, and then later, after delivery, blocking the response to the request, the client is not made aware of the delayed delivery except by the missing response. The server has in general, no way of knowing that the request was delayed and will therefore happily process the request.

If some wireless low-level protocol is used, the attack can also be performed by the attacker simultaneously recording what the client transmits while at the same time jamming the server. The request delay attack is illustrated in Figure 3.

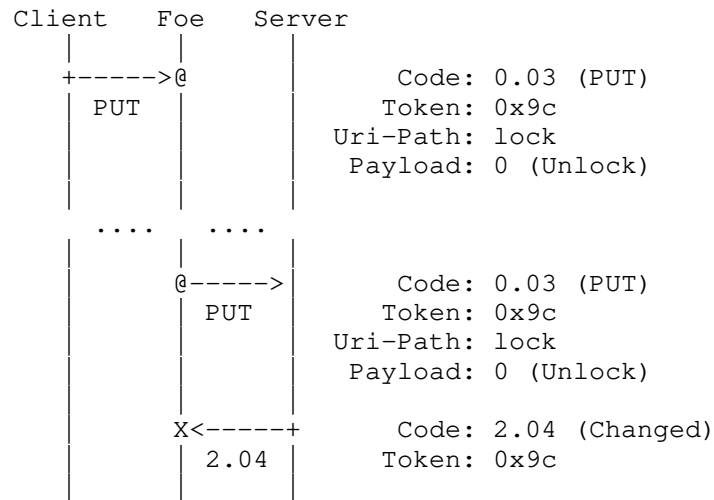


Figure 3: Delaying a Request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in time).

While an attacker delaying a request to a sensor is often not a security problem, an attacker delaying a request to an actuator performing an action is often a serious problem. A request to an actuator (for example a request to unlock a lock) is often only meant to be valid for a short time frame, and if the request does not reach the actuator during this short timeframe, the request should not be fulfilled. In the unlock example, if the client does not get any response and does not physically see the lock opening, the user is likely to walk away, calling the locksmith (or the IT-support).

If a non-zero replay window is used (the default when CoAP is used over UDP), the attacker can let the client interact with the actuator before delivering the delayed request to the server (illustrated in Figure 4). In the lock example, the attacker may store the first "unlock" request for later use. The client will likely resend the request with the same token. If DTLS is used, the resent packet will have a different sequence number and the attacker can forward it. If OSCOAP is used, resent packets will have the same sequence number and the attacker must block them all until the client sends a new message

with a new sequence number (not shown in Figure 4). After a while when the client has locked the door again, the attacker can deliver the delayed "unlock" message to the door, a very serious attack.

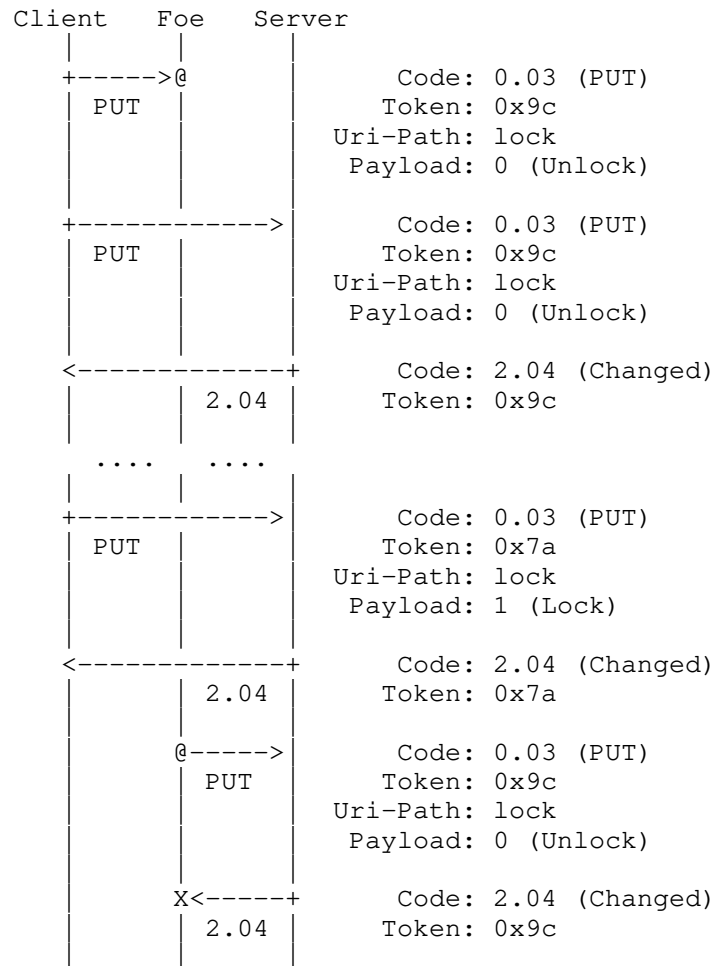


Figure 4: Delaying Request with Reordering

While the second attack (Figure 4) can be mitigated by using a replay window of length zero, the first attack (Figure 3) cannot. A solution must enable the server to verify that the request was received within a certain time frame after it was sent. This can be accomplished with either a challenge-response pattern, by exchanging timestamps, or by only allowing requests a short period after client authentication. Requiring a fresh client authentication (such as a new TLS/DTLS handshake or an EDHOC key exchange

[I-D.selander-ace-cose-ecdhe]) mitigates the problem, but requires larger messages and more processing than a dedicated solution. Security solutions based on timestamps require exactly synchronized time, and this is hard to control with complications such as time zones and daylight saving. Even if the clocks are synchronized at one point in time, they may easily get out-of-sync and an attacker may even be able to affect the client or the server time in various ways such as setting up a fake NTP server, broadcasting false time signals to radio controlled clocks, or expose one of them to a strong gravity field. As soon as client falsely believes it is time synchronized with the server, delay attacks are possible. A challenge response mechanism is much more failure proof and easy to analyze. The challenge and response may be sent in a CoAP option or in the CoAP payload. One such mechanism, the CoAP Replay Option, is specified in Section 3.

Remedy: The CoAP Replay Option specified in Section 3 SHALL be used for controlling actuators unless another application specific challenge-response or timestamp mechanism is used.

2.3. The Response Delay and Mismatch Attack

The following attack can be performed if CoAP is protected by a security protocol where the response is not bound to the request in any way except by the CoAP token. This would include most general security protocols, such as DTLS and IPsec, but not OSCOAP. The attacker performs the attack by delaying delivery of a response until the client sends a request with the same token. As long as the response is inside the replay window (which the attacker can make sure by blocking later responses), the response will be accepted by the client as a valid response to the later request. CoAP [RFC7252] uses a client generated token that the server echoes to match responses to request, but does not give any guidelines for the use of token with DTLS, except that the tokens currently "in use" SHOULD (not SHALL) be unique.

The attack can be performed by an attacker on the wire, or an attacker simultaneously recording what the server transmits while at the same time jamming the client. The response delay and mismatch attack is illustrated in Figure 5.

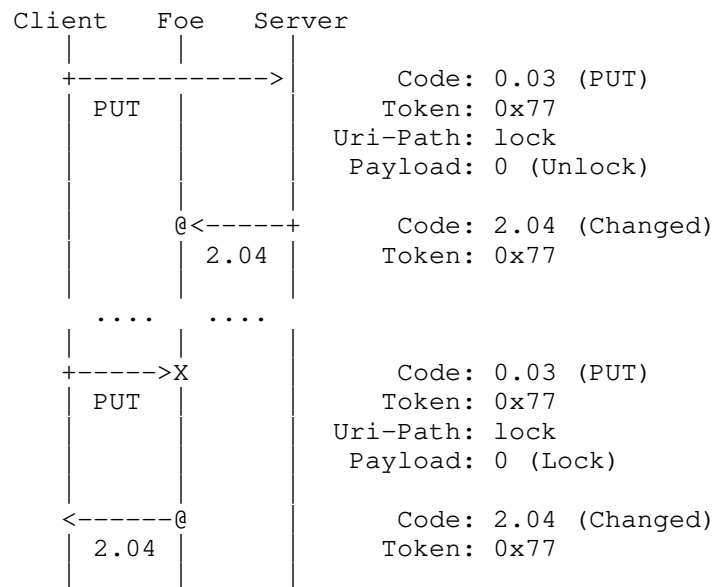


Figure 5: Delaying and Mismatching Response to PUT

If we once again take a lock as an example, the security consequences may be severe as the client receives a response message likely to be interpreted as confirmation of a locked door, while the received response message is in fact confirming an earlier unlock of the door. As the client is likely to leave the (believed to be locked) door unattended, the attacker may enter the home, enterprise, or car protected by the lock.

The same attack may be performed on sensors, also this with serious consequences. As illustrated in Figure 6, an attacker may convince the client that the lock is locked, when it in fact is not. The "Unlock" request may be also be sent by another client authorized to control the lock.

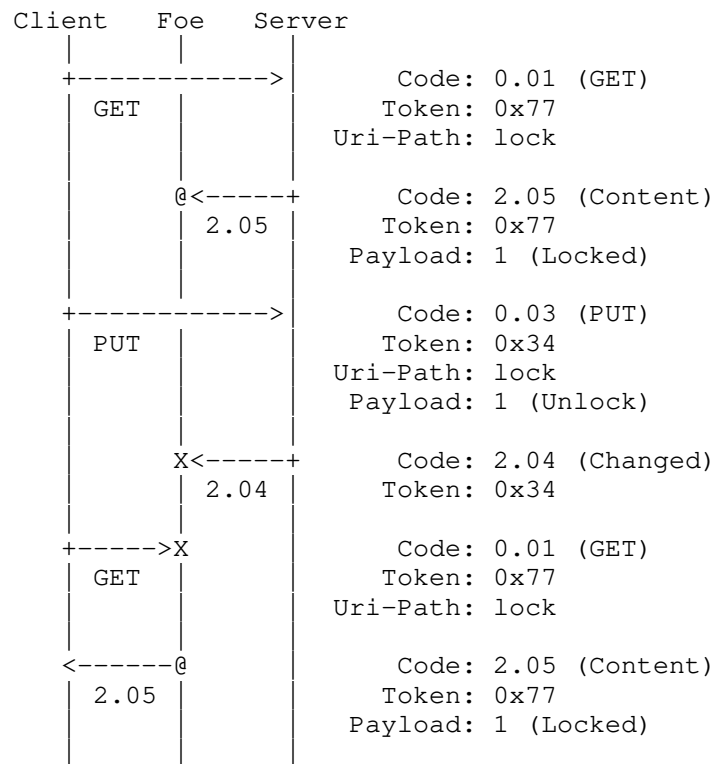


Figure 6: Delaying and Mismatching Response to GET

As illustrated in Figure 7, an attacker may even mix responses from different resources as long as the two resources share the same DTLS connection on some part of the path towards the client. This can happen if the resources are located behind a common gateway, or are served by the same CoAP proxy. An on-path attacker (not necessarily a DTLS endpoint such as a proxy) may e.g. deceive a client that the living room is on fire by responding with an earlier delayed response from the oven (temperatures in degree Celsius).

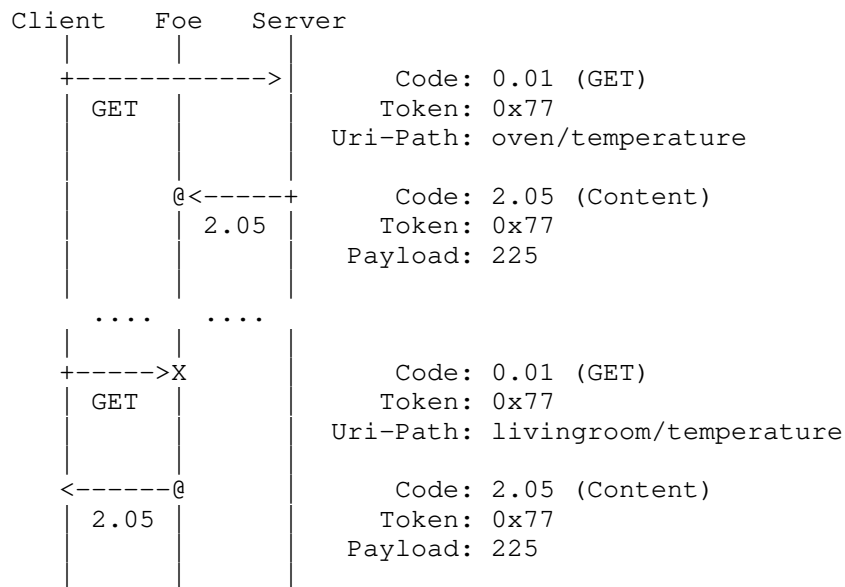


Figure 7: Delaying and Mismatching Response from other resource

Remedy: If CoAP is protected with a security protocol not providing bindings between requests and responses (e.g. DTLS) the client **MUST NOT** reuse any tokens for a given source/destination which the client has not received responses to. The easiest way to accomplish this is to implement the token as a counter and never reuse any tokens at all, this approach **SHOULD** be followed.

2.4. The Relay Attack

Yet another type of attack can be performed in deployments where actuator actions are triggered automatically based on proximity and without any user interaction, e.g. a car (the client) constantly polling for the car key (the server) and unlocking both doors and engine as soon as the car key responds. An attacker (or pair of attackers) may simply relay the CoAP messages out-of-band, using for examples some other radio technology. By doing this, the actuator (i.e. the car) believes that the client is close by and performs actions based on that false assumption. The attack is illustrated in Figure 8. In this example the car is using an application specific challenge-response mechanism transferred as CoAP payloads.

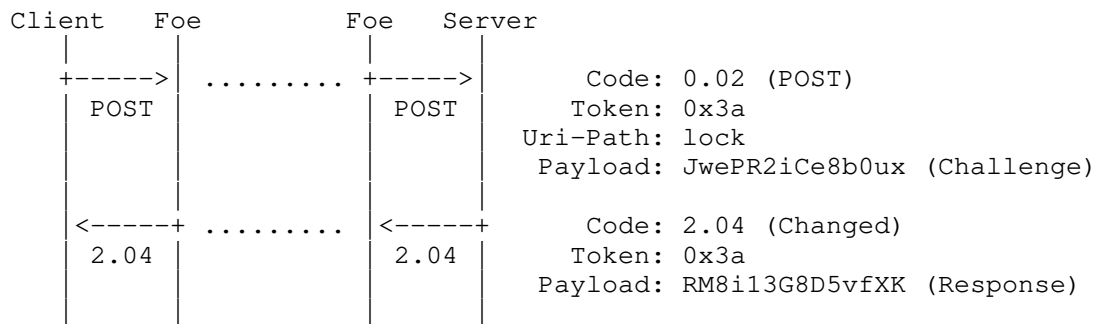


Figure 8: Relay Attack (the client is the actuator)

The consequences may be severe, and in the case of a car, lead to the attacker unlocking and driving away with the car, an attack that unfortunately is happening in practice.

Remedy: Getting a response over a short-range radio MUST NOT be taken as proof of proximity and therefore MUST NOT be used to take actions based on such proximity. Any automatically triggered mechanisms relying on proximity MUST use other stronger mechanisms to guarantee proximity. Mechanisms that MAY be used are: measuring the round-trip time and calculate the maximum possible distance based on the speed of light, or using radio with an extremely short range like NFC (centimeters instead of meters). Another option is to including geographical coordinates (from e.g. GPS) in the messages and calculate proximity based on these, but in this case the location measurements MUST be very precise and the system MUST make sure that an attacker cannot influence the location estimation, something that is very hard in practice.

3. The Repeat Option

The Repeat Option is a challenge-response mechanism for CoAP, binding a resent request to an earlier 4.03 forbidden response. The challenge (for the client) is simply to echo the Repeat Option value in a new request. The Repeat Option enables the server to verify the freshness of a request, thus mitigating the Delay Attack described in Section 2.2. An example message flow is illustrated in Figure 9.

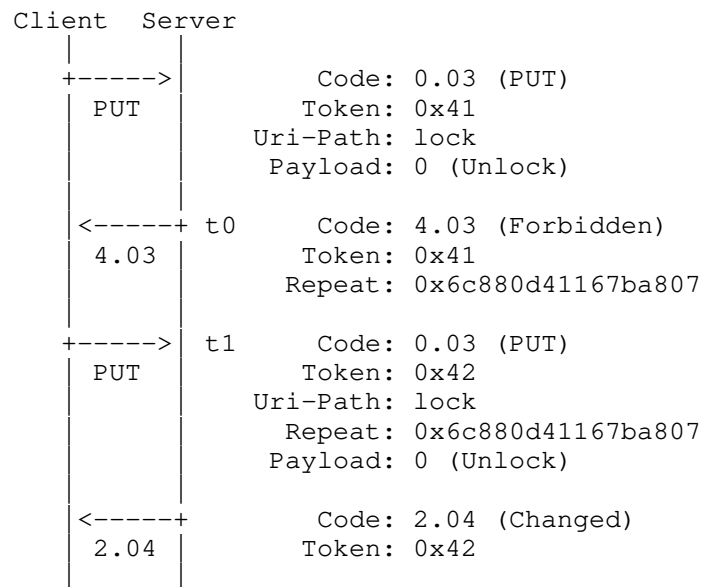


Figure 9: The Repeat Option

The Repeat Option may be used for all Methods and Response Codes. In responses, the value MUST be a (pseudo-)random bit string with a length of at least 64 bits. A new (pseudo-)random bit string MUST be generated for each response. In requests, the Repeat Option MUST echo the value from a previously received response.

The Repeat Option is critical, Safe-to-Forward, not part of the Cache-Key, and not repeatable.

Upon receiving a request without the Repeat Option to a resource with freshness requirements, the server sends a 4.03 Forbidden response with a Repeat Option and stores the option value and the response transmit time t_0 .

Upon receiving a 4.03 Forbidden response with the Repeat Option, the client SHOULD resend the request, echoing the Repeat Option value.

Upon receiving a request with the Repeat Option, the server verifies that the option value equals the previously sent value; otherwise the request is not processed further. The server calculates the round-trip time $RTT = (t_1 - t_0)$, where t_1 is the request receive time. The server MUST only accept requests with a round-trip time below a certain threshold T , i.e. $RTT < T$, otherwise the request is not processed further, and an error message MAY be send. The threshold T is application specific.

EDITORS NOTE: The mechanism described above is secure and gives the server freshness guarantee independently of what the client does. The disadvantages are that the mechanism always takes two round-trips and that the server has to save the option value and the time t_0 . Two different solutions involving time overcomes these disadvantages:

- o The server may simply send the client the current time in its timescale, i.e. a timestamp (option value = t_0). The client may then use this timestamp to estimate the current time in the servers timescale when sending future requests (i.e. not echoing). This approach has the benefit of reducing round-trips and server state, but has the security problems discussed in Section 2.2.
- o The server may instead of a pseudorandom value send an encrypted timestamp (option value = $E(k, t_0)$). CTR-mode would from a security point be like sending (value = t_0). ECB-mode or CCM-mode would work, but would expand the value length. With CCM, the server might also bind the option value to request (value = $AEAD(k, t_0, \text{parts of request})$). This approach does not reduce the number of round-trips but eliminates server state.

TODO: Update the Repeat Option to use a combination of these two solutions instead.

4. IANA Considerations

This document defines the following Option Number, whose value have been assigned to the CoAP Option Numbers Registry defined by [RFC7252].

Number	Name
29	Repeat

5. Security Considerations

The whole document can be seen as security considerations for CoAP.

6. Acknowledgements

The authors would like to thank Carsten Bormann, Klaus Hartke, Ari Keranen, Matthias Kovatsch, Sandeep Kumar, and Andras Mehes for their valuable comments and feedback.

7. References

7.1. Normative References

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

7.2. Informative References

- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Technologies, Z., and H. Tschofenig, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-tcp-tls-01 (work in progress), November 2015.
- [I-D.ietf-cose-msg]
Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-10 (work in progress), February 2016.
- [I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-00 (work in progress), March 2016.
- [I-D.selander-ace-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-03 (work in progress), October 2015.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.

Authors' Addresses

John Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

John Fornehed
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.fornehed@ericsson.com

Goran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 21, 2019

J. Mattsson
J. Fornehed
G. Selander
F. Palombini
Ericsson
C. Amsuess
Energy Harvesting Solutions
September 17, 2018

Controlling Actuators with CoAP
draft-mattsson-core-coap-actuators-06

Abstract

Being able to trust information from sensors and to securely control actuators are essential in a world of connected and networking things interacting with the physical world. In this memo we show that just using COAP with a security protocol like DTLS, TLS, or OSCORE is not enough. We describe several serious attacks any on-path attacker can do, and discusses tougher requirements and mechanisms to mitigate the attacks. While this document is focused on actuators, some of the attacks apply equally well to sensors.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 21, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. Attacks	4
2.1. The Block Attack	4
2.2. The Request Delay Attack	5
2.3. The Response Delay and Mismatch Attack	8
2.4. The Relay Attack	11
2.5. The Request Fragment Rearrangement Attack	12
2.5.1. Completing an Operation with an Earlier Final Block .	13
2.5.2. Injecting a Withheld First Block	14
3. Security Considerations	15
4. IANA Considerations	15
5. References	15
5.1. Normative References	15
5.2. Informative References	16
Acknowledgements	17
Authors' Addresses	17

1. Introduction

Being able to trust information from sensors and to securely control actuators are essential in a world of connected and networking things interacting with the physical world. One protocol used to interact with sensors and actuators is the Constrained Application Protocol (CoAP) [RFC7252]. Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [RFC6347], TLS [RFC5246], or OSCORE [I-D.ietf-core-object-security] to protect CoAP, where the choice of security protocol depends on the transport protocol and the presence of intermediaries. The use of CoAP over UDP and DTLS is specified in [RFC6347] and the use of CoAP over TCP and TLS is specified in [RFC8323]. OSCORE protects CoAP end-to-end with the use of COSE [RFC8152] and the CoAP Object-Security option [I-D.ietf-core-object-security], and can therefore be used over any transport.

The Constrained Application Protocol (CoAP) [RFC7252] was designed with the assumption that security could be provided on a separate

layer, in particular by using DTLS [RFC6347]. The four properties traditionally provided by security protocols are:

- o Data confidentiality
- o Data origin authentication
- o Data integrity checking
- o Replay protection

In this document we show that protecting CoAP with a security protocol on another layer is not nearly enough to securely control actuators (and in many cases sensors) and that secure operation often demands far more than the four properties traditionally provided by security protocols. We describe several serious attacks any on-path attacker (i.e. not only "trusted intermediaries) can do and discusses tougher requirements and mechanisms to mitigate the attacks. In general, secure operation of actuators also requires the three properties:

- o Data-to-Data binding
- o Data-to-space binding
- o Data-to-time binding

"Data-to-Data binding" is e.g. binding of responses to a request or binding of data fragments to each other. "Data-to-space binding" is the binding of data to an absolute or relative point in space (i.e. a location) and may in the relative case be referred to as proximity. "Data-to-time binding" is the binding of data to an absolute or relative point in time and may in the relative case be referred to as freshness. The two last properties may be bundled together as "Data-to-spacetime binding".

The request delay attack (valid for DTLS, TLS, and OSCORE and described in Section 2.2) lets an attacker control an actuator at a much later time than the client anticipated. The response delay and mismatch attack (valid for DTLS and TLS and described in Section 2.3) lets an attacker respond to a client with a response meant for an older request. The request fragment rearrangement attack (valid for DTLS, TLS, and OSCORE and described in Section 2.5) lets an attacker cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations.

Mechanisms mitigating some of the attacks discussed in this document can be found in [I-D.ietf-core-echo-request-tag] and [I-D.liu-core-coap-delay-attacks]

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Attacks

Internet-of-Things (IoT) deployments valuing security and privacy, MUST use a security protocol such as DTLS, TLS, or OSCORE to protect CoAP. This is especially true for deployments of actuators where attacks often (but not always) have serious consequences. The attacks described in this section are made under the assumption that CoAP is already protected with a security protocol such as DTLS, TLS, or OSCORE, as an attacker otherwise can easily forge false requests and responses.

2.1. The Block Attack

An on-path attacker can block the delivery of any number of requests or responses. The attack can also be performed by an attacker jamming the lower layer radio protocol. This is true even if a security protocol like DTLS, TLS, or OSCORE is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS and TLS, proxies have access to the complete CoAP message, and with OSCORE, the CoAP header and several CoAP options are not encrypted. In both security protocols, the IP-addresses, ports, and CoAP message lengths are available to all on-path attackers, which may be enough to determine the server, resource, and command. The block attack is illustrated in Figures 1 and 2.

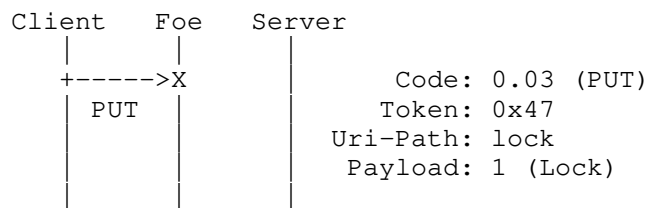


Figure 1: Blocking a request

Where 'X' means the attacker is blocking delivery of the message.

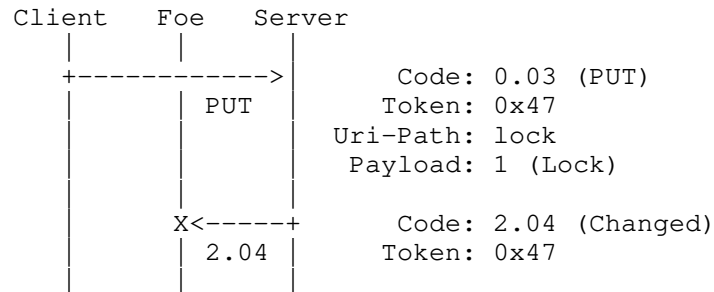


Figure 2: Blocking a response

While blocking requests to, or responses from, a sensor is just a denial of service attack, blocking a request to, or a response from, an actuator results in the client losing information about the server's status. If the actuator e.g. is a lock (door, car, etc.), the attack results in the client not knowing (except by using out-of-band information) whether the lock is unlocked or locked, just like the observer in the famous Schrodinger's cat thought experiment. Due to the nature of the attack, the client cannot distinguish the attack from connectivity problems, offline servers, or unexpected behavior from middle boxes such as NATs and firewalls.

Remedy: Any IoT deployment of actuators where confirmation is important MUST notify the user upon reception of the response, or warn the user when a response is not received.

2.2. The Request Delay Attack

An on-path attacker may not only block packets, but can also delay the delivery of any packet (request or response) by a chosen amount of time. If CoAP is used over a reliable and ordered transport such as TCP with TLS or OSCORE, no messages can be delivered before the delayed message. If CoAP is used over an unreliable and unordered transport such as UDP with DTLS, or OSCORE, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. When CoAP is used over UDP, both DTLS and OSCORE allow out-of-order delivery and uses sequence numbers together with a replay window to protect against replay attacks. The replay window has a default length of 64 in DTLS and 32 in OSCORE. The attacker can control the replay window by blocking some or all other packets. By first delaying a request, and then later, after delivery, blocking the response to the request, the client is not made aware of the delayed delivery except by the missing response. The server has in general, no way of knowing that the request was

delayed and will therefore happily process the request. Note that delays can also happen for other reasons than a malicious attacker.

If some wireless low-level protocol is used, the attack can also be performed by the attacker simultaneously recording what the client transmits while at the same time jamming the server. The request delay attack is illustrated in Figure 3.

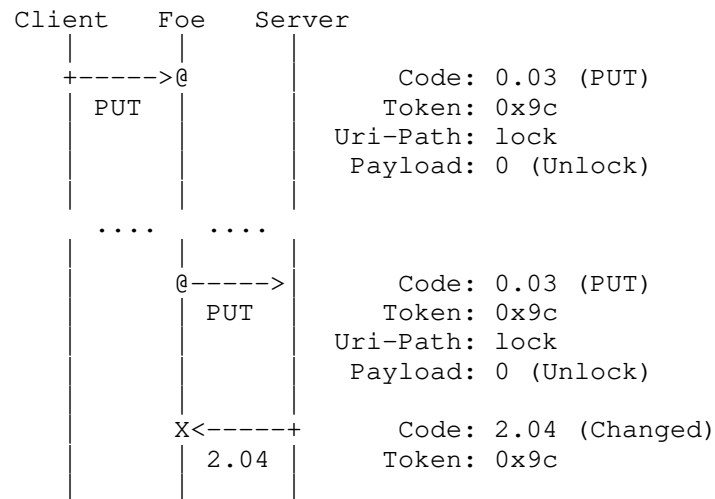


Figure 3: Delaying a request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in time).

While an attacker delaying a request to a sensor is often not a security problem, an attacker delaying a request to an actuator performing an action is often a serious problem. A request to an actuator (for example a request to unlock a lock) is often only meant to be valid for a short time frame, and if the request does not reach the actuator during this short timeframe, the request should not be fulfilled. In the unlock example, if the client does not get any response and does not physically see the lock opening, the user is likely to walk away, calling the locksmith (or the IT-support).

If a non-zero replay window is used (the default when CoAP is used over UDP), the attacker can let the client interact with the actuator before delivering the delayed request to the server (illustrated in Figure 4). In the lock example, the attacker may store the first "unlock" request for later use. The client will likely resend the request with the same token. If DTLS is used, the resent packet will

have a different sequence number and the attacker can forward it. If OSCORE is used, resent packets will have the same sequence number and the attacker must block them all until the client sends a new message with a new sequence number (not shown in Figure 4). After a while when the client has locked the door again, the attacker can deliver the delayed "unlock" message to the door, a very serious attack.

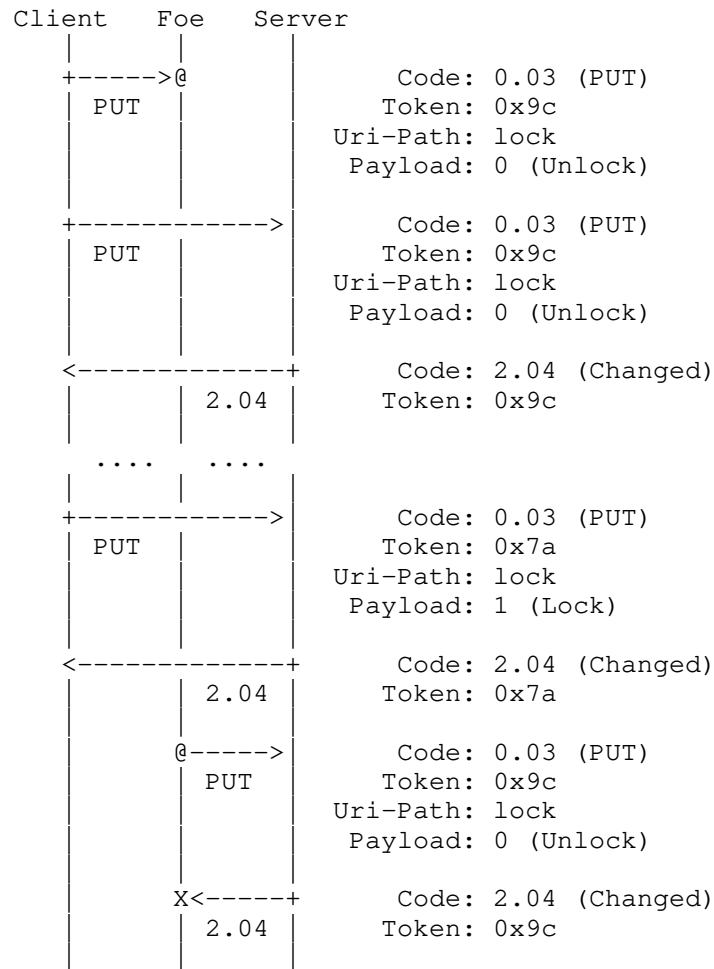


Figure 4: Delaying request with reordering

While the second attack (Figure 4) can be mitigated by using a replay window of length zero, the first attack (Figure 3) cannot. A solution must enable the server to verify that the request was received within a certain time frame after it was sent or enable the server to securely determine an absolute point in time when the

request is to be executed. This can be accomplished with either a challenge-response pattern, by exchanging timestamps between client and server, or by only allowing requests a short period after client authentication.

Requiring a fresh client authentication (such as a new TLS/DTLS handshake or an EDHOC key exchange [I-D.selander-ace-cose-ecdhe]) mitigates the problem, but requires larger messages and more processing than a dedicated solution. Security solutions based on exchanging timestamps require exactly synchronized time between client and server, and this may be hard to control with complications such as time zones and daylight saving. Wall clock time SHOULD NOT be used as it is not monotonic, may reveal that the endpoints will accept expired certificates, or reveal the endpoint's location. Use of non-monotonic clocks is not secure as the server will accept requests if the clock is moved backward and reject requests if the clock is moved forward. Even if the clocks are synchronized at one point in time, they may easily get out-of-sync and an attacker may even be able to affect the client or the server time in various ways such as setting up a fake NTP server, broadcasting false time signals to radio controlled clocks, or expose one of them to a strong gravity field. As soon as client falsely believes it is time synchronized with the server, delay attacks are possible. A challenge response mechanism where the server does not need to synchronize its time with the client is easier to analyze but require more roundtrips. The challenges, responses, and timestamps may be sent in a CoAP option or in the CoAP payload.

Remedy: The mechanisms specified in [I-D.ietf-core-echo-request-tag] or [I-D.liu-core-coap-delay-attacks] SHALL be used for controlling actuators unless another application specific challenge-response or timestamp mechanism is used.

2.3. The Response Delay and Mismatch Attack

The following attack can be performed if CoAP is protected by a security protocol where the response is not bound to the request in any way except by the CoAP token. This would include most general security protocols, such as DTLS, TLS, and IPsec, but not OSCORE. CoAP [RFC7252] uses a client generated token that the server echoes to match responses to request, but does not give any guidelines for the use of token with DTLS and TLS, except that the tokens currently "in use" SHOULD (not SHALL) be unique. The attacker performs the attack by delaying delivery of a response until the client sends a request with the same token, the response will be accepted by the client as a valid response to the later request. If CoAP is used over a reliable and ordered transport such as TCP with TLS, no messages can be delivered before the delayed message. If CoAP is

used over an unreliable and unordered transport such as UDP with DTLS, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. Note that mismatches can also happen for other reasons than a malicious attacker, e.g. delayed delivery or a server sending notifications to an uninterested client.

The attack can be performed by an attacker on the wire, or an attacker simultaneously recording what the server transmits while at the same time jamming the client. The response delay and mismatch attack is illustrated in Figure 5.

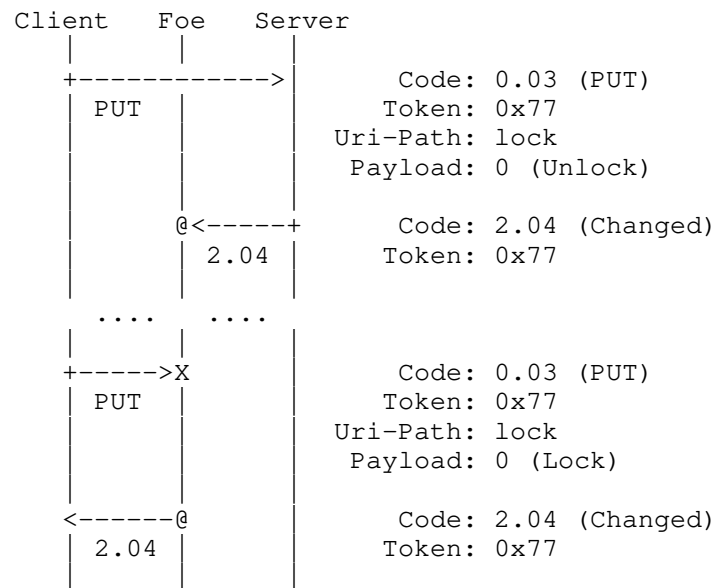


Figure 5: Delaying and mismatching response to PUT

If we once again take a lock as an example, the security consequences may be severe as the client receives a response message likely to be interpreted as confirmation of a locked door, while the received response message is in fact confirming an earlier unlock of the door. As the client is likely to leave the (believed to be locked) door unattended, the attacker may enter the home, enterprise, or car protected by the lock.

The same attack may be performed on sensors, also this with serious consequences. As illustrated in Figure 6, an attacker may convince the client that the lock is locked, when it in fact is not. The "Unlock" request may be also be sent by another client authorized to control the lock.

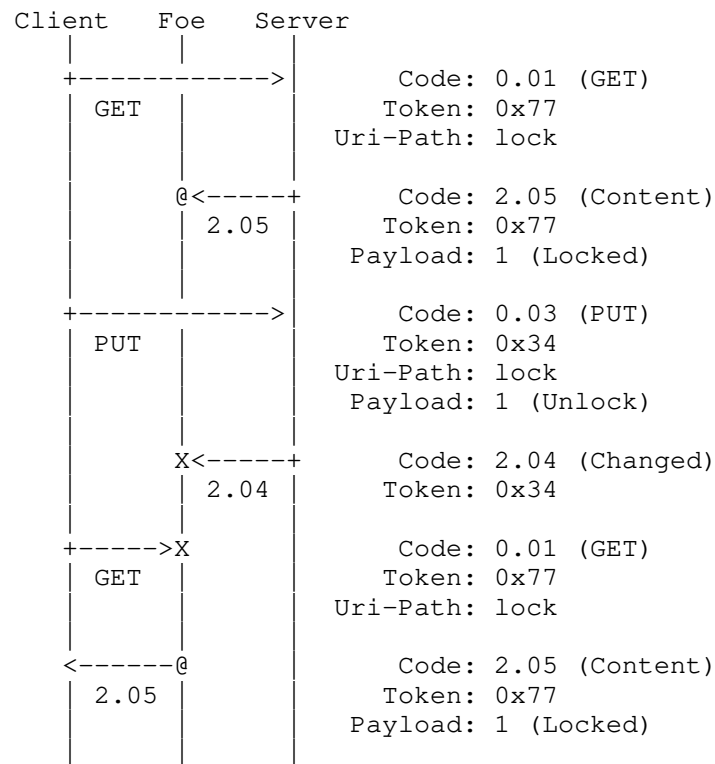


Figure 6: Delaying and mismatching response to GET

As illustrated in Figure 7, an attacker may even mix responses from different resources as long as the two resources share the same (D)TLS connection on some part of the path towards the client. This can happen if the resources are located behind a common gateway, or are served by the same CoAP proxy. An on-path attacker (not necessarily a (D)TLS endpoint such as a proxy) may e.g. deceive a client that the living room is on fire by responding with an earlier delayed response from the oven (temperatures in degree Celsius).

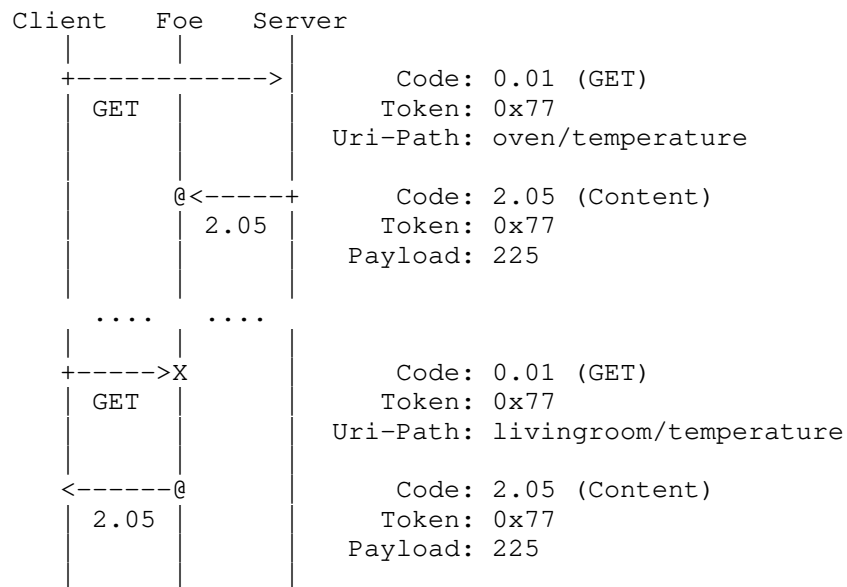


Figure 7: Delaying and mismatching response from other resource

Remedy: If CoAP is protected with a security protocol not providing bindings between requests and responses (e.g. DTLS and TLS) the client MUST NOT reuse any tokens until the traffic keys have been replaced. The easiest way to accomplish this is to implement the Token as a counter, this approach SHOULD be followed.

2.4. The Relay Attack

Yet another type of attack can be performed in deployments where actuator actions are triggered automatically based on proximity and without any user interaction, e.g. a car (the client) constantly polling for the car key (the server) and unlocking both doors and engine as soon as the car key responds. An attacker (or pair of attackers) may simply relay the CoAP messages out-of-band, using for examples some other radio technology. By doing this, the actuator (i.e. the car) believes that the client is close by and performs actions based on that false assumption. The attack is illustrated in Figure 8. In this example the car is using an application specific challenge-response mechanism transferred as CoAP payloads.

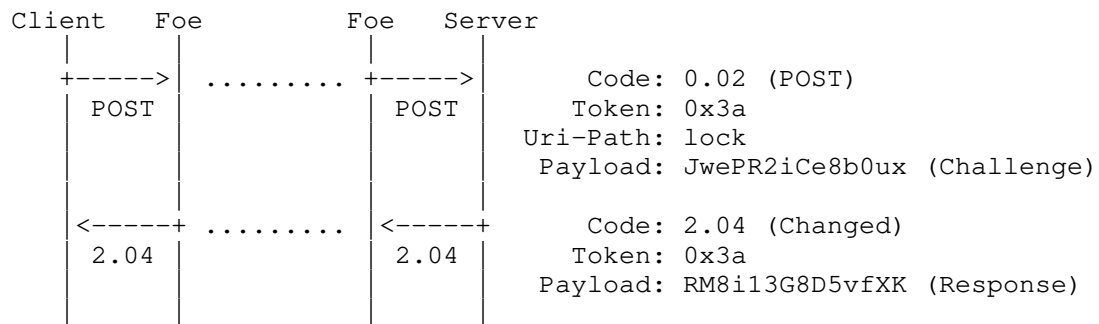


Figure 8: Relay attack (the client is the actuator)

The consequences may be severe, and in the case of a car, lead to the attacker unlocking and driving away with the car, an attack that unfortunately is happening in practice.

Remedy: Getting a response over a short-range radio MUST NOT be taken as proof of proximity and therefore MUST NOT be used to take actions based on such proximity. Any automatically triggered mechanisms relying on proximity MUST use other stronger mechanisms to guarantee proximity. Mechanisms that MAY be used are: measuring the round-trip time and calculate the maximum possible distance based on the speed of light, or using radio with an extremely short range like NFC (centimeters instead of meters) that cannot be relayed through e.g. clothes. Another option is to including geographical coordinates (from e.g. GPS) in the messages and calculate proximity based on these, but in this case the location measurements MUST be very precise and the system MUST make sure that an attacker cannot influence the location estimation, something that is very hard in practice.

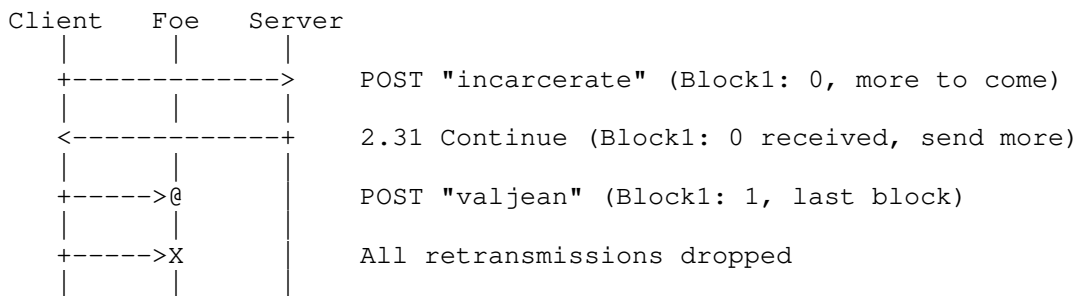
2.5. The Request Fragment Rearrangement Attack

These attack scenarios show that the Request Delay and Block Attacks can be used against blockwise transfers to cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations. The combination of these attacks is described as a separate attack because it makes the Request Delay Attack relevant to systems that are otherwise not time-dependent, which means that they could disregard the Request Delay Attack.

This attack works even if the individual request/response pairs are encrypted, authenticated and protected against the Response Delay and Mismatch Attack, provided the attacker is on the network path and can correctly guess which operations the respective packages belong to.

2.5.1. Completing an Operation with an Earlier Final Block

In this scenario (illustrated in Figure 9), blocks from two operations on a POST-accepting resource are combined to make the server execute an action that was not intended by the authorized client. This works only if the client attempts a second operation after the first operation failed (due to what the attacker made appear like a network outage) within the replay window. The client does not receive a confirmation on the second operation either, but, by the time the client acts on it, the server has already executed the unauthorized action.



(Client: Odd, but let's go on and promote Javert)

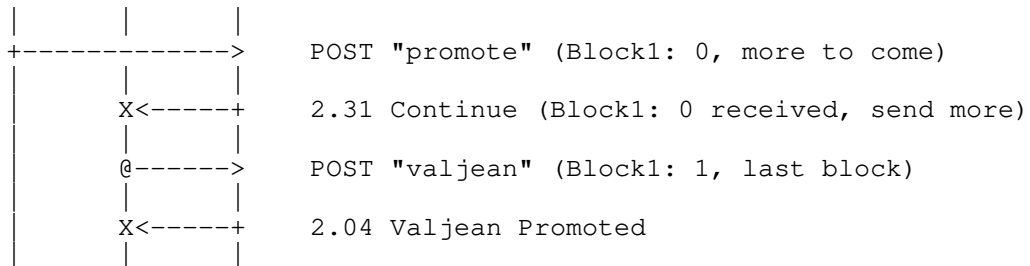


Figure 9: Completing an operation with an earlier final block

Remedy: If a client starts new blockwise operations on a security context that has lost packages, it needs to label the fragments in such a way that the server will not mix them up.

A mechanism to that effect is described as Request-Tag [I-D.ietf-core-echo-request-tag]. Had it been in place in the example and used for body integrity protection, the client would have set the Request-Tag option in the "promote" request. Depending on the server's capabilities and setup, either of four outcomes could have occurred:

1. The server could have processed the reinjected POST "valjean" as belonging to the original "incarcerate" block; that's the expected case when the server can handle simultaneous block transfers.
2. The server could respond 5.03 Service Unavailable, including a Max-Age option indicating how long it prefers not to take any requests that force it to overwrite the state kept for the "incarcerate" request.
3. The server could decide to drop the state kept for the "incarcerate" request's state, and process the "promote" request. The reinjected POST "valjean" will then fail with 4.08 Request Entity incomplete, indicating that the server does not have the start of the operation any more.

2.5.2. Injecting a Withheld First Block

If the first block of a request is withheld by the attacker for later use, it can be used to have the server process a different request body than intended by the client. Unlike in the previous scenario, it will return a response based on that body to the client.

Again, a first operation (that would go like "Homeless stole apples. What shall we do with him?" - "Set him free.") is aborted by the proxy, and a part of that operation is later used in a different operation to prime the server for responding leniently to another operation that would originally have been "Hitman killed someone. What shall we do with him?" - "Hang him.". The attack is illustrated in Figure 10.

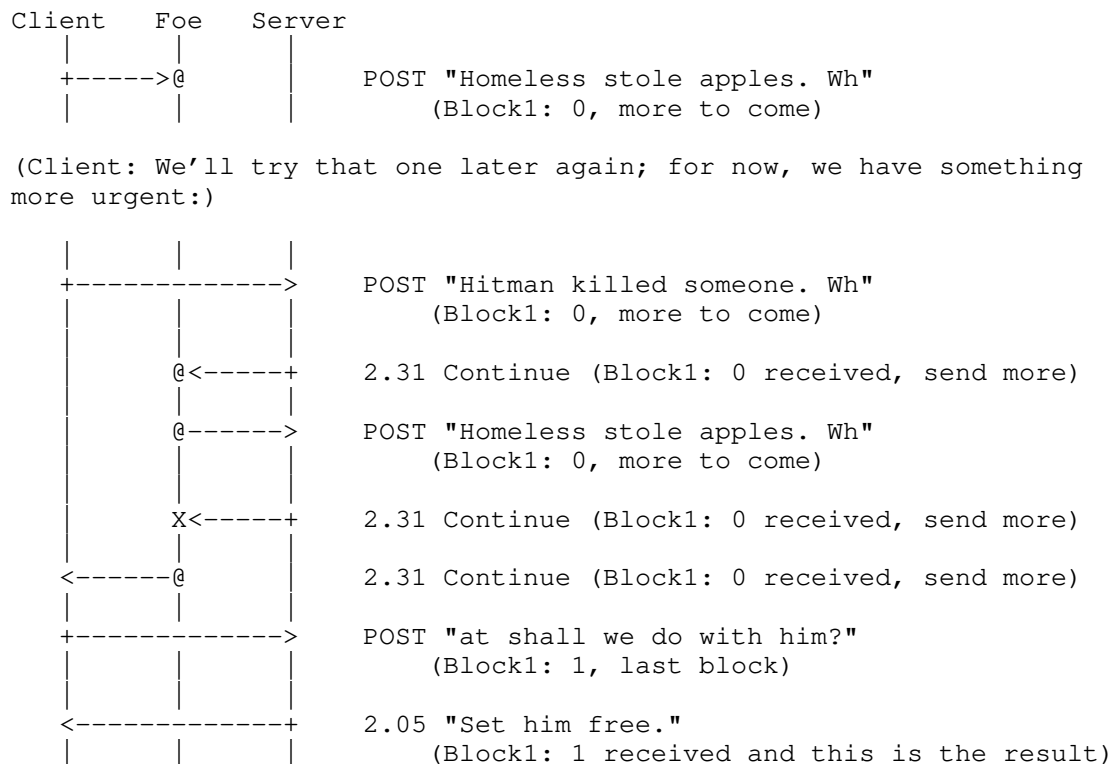


Figure 10: Injecting a withheld first block

3. Security Considerations

The whole document can be seen as security considerations for CoAP.

4. IANA Considerations

This document has no actions for IANA.

5. References

5.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

5.2. Informative References

- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-02 (work in progress), June 2018.
- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-15 (work in progress), August 2018.
- [I-D.liu-core-coap-delay-attacks]
Liu, Y. and J. Zhu, "Mitigating delay attacks on Constrained Application Protocol", draft-liu-core-coap-delay-attacks-01 (work in progress), October 2017.
- [I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-09 (work in progress), July 2018.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <<https://www.rfc-editor.org/info/rfc8323>>.

Acknowledgements

The authors would like to thank Carsten Bormann, Klaus Hartke, Ari Keraenen, Matthias Kovatsch, Sandeep Kumar, and Andras Mehes for their valuable comments and feedback.

Authors' Addresses

John Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

John Fornehed
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.fornehed@ericsson.com

Goeran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Christian Amsuess
Energy Harvesting Solutions

Email: c.amsuess@energyharvesting.at

Internet Engineering Task Force
Internet-Draft

Intended status: Informational

Expires: August 20, 2016

A. Pelov, Ed.

Acklio

L. Toutain, Ed.

Institut MINES-TELECOM ; TELECOM Bretagne

Y. Delibie, Ed.

Kerlink

February 17, 2016

Constrained Signaling Over LP-WAN
draft-pelov-core-cosol-01

Abstract

This document presents a new type of long-range, low-rate radio technologies and an extensible mechanism to operate these networks based on CoAP. The emerging Low-Power Wide-Area Networks (LP-WAN) present a particular set of constraints, which places them at the intersection of infrastructure networks, ultra-dense networks, delay-tolerant networks and low-power and lossy networks. The main objectives of LP-WAN signaling is to minimize the number of exchanged messages, minimize the size of each message in a secure and extensible manner, all with keeping the fundamental principle of technology-independence (L2-independence). This document describes the use of the Constrained Application Protocol (CoAP) as the main signaling protocol for LP-WAN, over which minimal messages are exchanged allowing the full operation of the network, such as authentication, authorization, and management. The use of CoAP signaling provides a generic mechanism that can be applied to different LP-WAN technologies.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 20, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	5
2. LP-WAN Technologies	5
2.1. Radio technologies	5
2.2. Physical Layer Characteristics	5
2.2.1. Ultra Narrowband LP-WAN radios	6
2.2.2. Spread-spectrum LP-WAN radios	6
2.3. MAC Layer Characteristics	6
3. CoSOL Architecture	7
3.1. General LP-WAN architecture	7
3.2. Node-F lifecycle	8
3.3. CoAP as Signaling Protocol for LP-WANs	10
3.3.1. Semi-Association	10
3.3.2. Network Discovery	11
3.3.3. Association	13
3.3.4. Authentication	13
3.3.5. Operation	14
3.3.6. Dissociation	15
4. Acknowledgements	15
5. IANA Considerations	15
6. Security Considerations	16
7. References	16
7.1. Normative References	16
7.2. Informative References	17
Authors' Addresses	17

1. Introduction

The goal of this document is to provide the necessary mechanisms to operate a Low-Power Wide-Area Network (LP-WAN) by using IETF CoAP [RFC7252] as a core signaling protocol.

Long-range, low-rate radio technologies have emerged in the past several years, and are the base for building LP-WANs. LP-WANs generally have the following characteristics:

- o Work in narrow, license-free (ISM) bands with good propagation properties (< 1GHz)
- o Low- to very-low throughput (1-200 kbps)
- o Low-power operation (25 mW in Europe)
- o Long-range communication capabilities (up to 30 km with line-of-sight, several km in urban environment)
- o Strong channel access restrictions (1% to 10% duty cycling)
- o Infrastructure-based
- o Star topology

LP-WANs are built on radio communication technologies, which use advanced signal processing techniques and combination of appropriate modulation and coding approaches to provide the aforementioned radio characteristics.

The absence of license fees and the far-reaching connectivity allow for an extremely competitive pricing of LP-WANs compared to other networking technologies, e.g. cellular or mesh. LP-WANs are sometimes referred to as LPWA or LR-WAN (Low-Rate WAN). Even though LP-WANs are extremely limited in terms of network performance, they are enough for a wide class of applications, among which [LTN001]:

- o Metering (water, gas, electricity)
- o Infrastructure networks (water, gas, electricity, roads, pipelines, drains)
- o Environment/Smart City (waste management, air pollution monitoring and alerting, acoustic noise monitoring, public lighting management, parking management, self service bike rental, digital board monitoring, water pipe leakage monitoring)
- o Environment/Country side (soil quality, livestock surveillance, cattle and pet monitoring, climate, irrigation)
- o Remote monitoring (house, building)
- o Industrial (water tank, asset tracking)

- o Automotive (vehicle tracking, impact detection, pay as you drive, assistance request, ...)
- o Logistics (goods tracking, conservation monitoring)
- o Healthcare (patient monitoring, home medical equipment usage)
- o House appliances (pet tracking, white goods, personal asset)
- o Truck (tyre monitoring)
- o Identification (authentication)

The IEEE is studying LP-WANs, but limited to the case of low-energy critical infrastructure monitoring (LECIM), under the group IEEE 802.15.4k [IEEE.802-15.4k].

The combination of the above characteristics and the envisioned applications define a new class of networks with the following unique constraints:

- o Potentially extremely high density (expected of up to 10k-100k+ end-devices managed by a single radio antenna)
- o Coexistence of delay-tolerant and critical applications (metering and alarms)
- o Low-power, low-throughput, lossy connectivity (use of ISM bands)
- o Limited payload (100 bytes max, typically less than 50 bytes, 12 bytes for UNB)

CoAP is a client-server protocol specialized for constrained networks and devices. CoAP is highly optimized, extensible, standard protocol, which in conjunction with the Concise Binary Object Representation (CBOR) is the ideal candidate for the signaling protocol of the control plane of an LP-WAN.

It can be used during all stages of the lifecycle of the network, e.g. discovery, authentication, operation. Furthermore, this can be achieved by following RESTful management paradigm, by using a particular resource tree definition or adopting COOL [I-D.veillette-core-cool].

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. LP-WAN Technologies

2.1. Radio technologies

There are two classes of LP-WAN radio technologies, using different radio modulation approaches:

- o Ultra Narrow Band (UNB)
- o Spread-spectrum (SS)

An example of UNB is the technology developed and promoted by SigFox [SigFox]. Semtech LoRa [LoRa] uses a direct-sequence spread-spectrum with orthogonal codes (OSSS).

Both approaches have their advantages and will coexist in the future, as there are currently several operators, which deploy the two types in the same areas.

2.2. Physical Layer Characteristics

At the physical layer, the important part is the possibility to reconstruct the signal at long distances. The used ISM bands are defined around the world (e.g. 868 MHz in Europe and 900 MHz in USA) and require a 1% (or 10%) duty cycling, or alternatively - advanced detection and channel reallocation techniques. In reality, all deployed networks use the duty cycling limitation, with the following distinction. There is one 100kHz band in which 10% duty cycling is allowed, with a slightly more emission power. The rest of the bands are limited at 1% duty cycling and very restricted power of emission (e.g. 25 mW in Europe).

UNB LP-WANs make the distinction between Uplink and Downlink, first depending on the modulation, and second with the 10% duty-cycling channel been used for the Downlink. OSSS LP-WANs make no such distinction, although for the operation of a network, an operator can chose to use the same Uplink/Downlink channel separation.

Note that the 1% or 10% duty-cycle limitation counts for all traffic originating from an electronic equipment, e.g. an antenna managing 100k objects must obey the same limitation as an end-device, with all

frames emitted from the antenna (data, acknowledgements) counting towards its quota.

2.2.1. Ultra Narrowband LP-WAN radios

Ultra Narrowband (UNB) technologies generally possess the following physical layer characteristics [LTN003]:

- o Uplink:

- * channelization mask 100kHz (600 kHz USA)
- * baud rate 100 bauds (600 bauds USA)
- * modulation BPSK

- o Downlink:

- * channelization mask: dynamic selection
- * down link baud rate: 600 baud
- * modulation scheme: GFSK
- * downlink transmission power: 500 mW, 10% duty cycle

2.2.2. Spread-spectrum LP-WAN radios

OSSS technologies possess the following physical layer characteristics [LTN003]:

- o channelization mask: from 8 kHz to 500 kHz (depending on spreading factor)
- o chip rate: 8 kcps up to 500 kcps
- o data rate: 30-50 000 bps
- o modulation scheme: equivalent to DSSS with orthogonal signaling

No particular distinction is made between the Uplink and the Downlink.

2.3. MAC Layer Characteristics

Several proprietary MAC frame formats exist for UNB and OSSS. However, they are designed to operate the network in a centralized, highly-vertically-integrated fashion. The only standard MAC frame

format is the IEEE 802.15.4k, which is based on the well-known IEEE 802.15.4 with the addition of a fragmentation sub-layer.

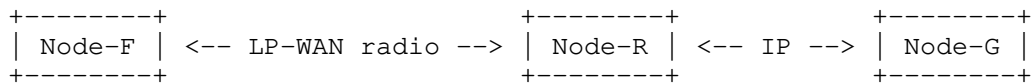
The channel access method is based on ALOHA, although it is up to the network operator to chose if an appropriate end-node polling should be implemented.

3. CoSOL Architecture

3.1. General LP-WAN architecture

We can identify three types of entities in a typical LP-WAN. These are:

- o Node-F: far-reachable node, e.g. the end-point, object, device.
- o Node-R: radio relay, bridging the LP-WAN radio technology to a different medium (often a LAN or cellular WAN).
- o Node-G: gateway node, interconnection between the radio-relay node and the Internet.



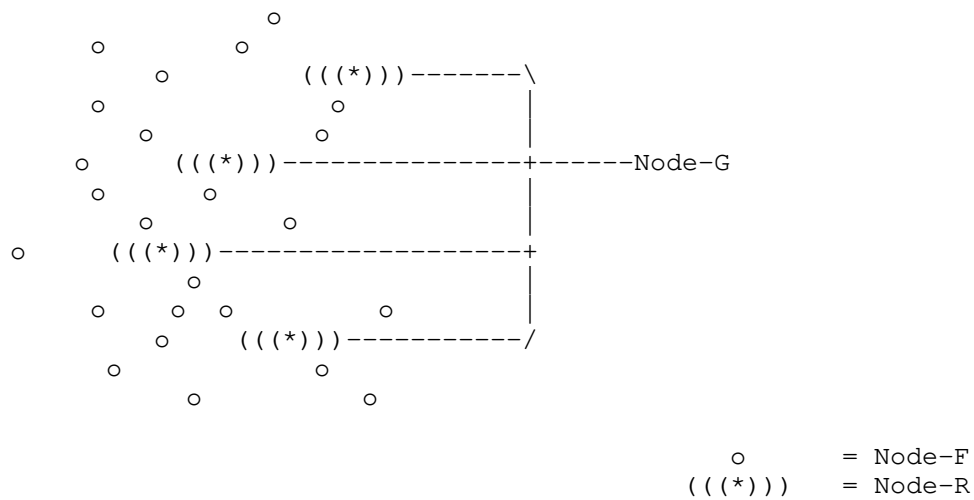
General architecture of an LP-WAN. LP-WAN radio technology is used only between the Node-F and the Node-R.

Figure 1

Of these, only Node-F and Node-R communicate through an LP-WAN radio technology. However, due to the extreme constraints of these technologies, they are always behind a gateway (Node-G). Note, that the Node-R and Node-G can be collocated, e.g. on a single hardware equipment.

The Node-G is connected to the Internet and is assumed to have sufficient computational resources to store a context for each of the Node-Fs. The strong limitation here is the radio link.

In an actual deployment, a (limited) set of Node-Rs cover a large area with a potentially very-high number of Node-Fs. A single Node-G is capable of controlling all Node-Rs.



An example coverage of an area with several Node-Rs. Note that a single Node-F may be covered by several Node-Rs.

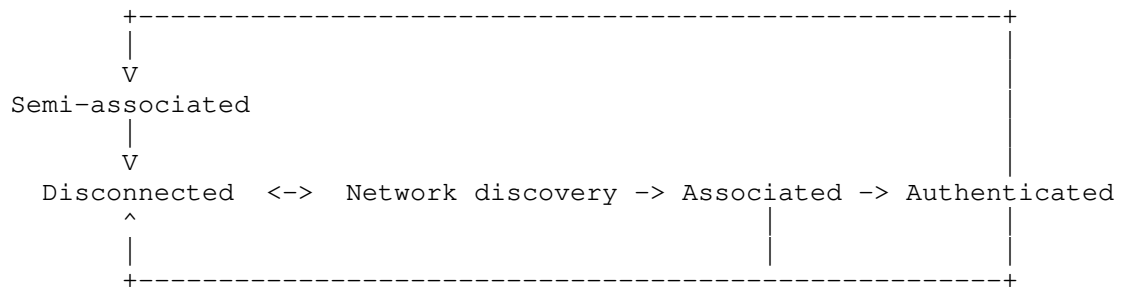
Figure 2

3.2. Node-F lifecycle

Similar to other wireless infrastructure-based technologies, a Node-F can go through several stages:

- Semi-Association
- Network Discovery
- Association
- Authentication
- Dissociation

The Node-F state machine is then the following:



Node-F connectivity state machine.

Figure 3

The Node-F can be in Semi-Associated mode. Upon start, and depending on the application, a Node-F can use a state of uni-directional communication, where it is considered semi-associated to the network. In that state, the Node-F broadcasts frames, handled by the Node-G, but the network cannot join the Node-F on a regular basis. This is a degraded LP-WAN operating mode and if caution is not used, can lead to significant scalability and evolvability issues.

The Network Discovery can be reactive or proactive. The former is based on detecting beacon frames sent periodically by the network (e.g. Node-G). The latter is implemented by the Node-F broadcasting probe request frames, to which all appropriate Node-Gs must respond.

Once a network has been discovered, the Node-F can associate to the network. The association creates the necessary (minimal) context on the Node-G, which initiates the authentication of the Node-F

The authentication is initiated by the Node-G, which should allow for the necessary AAA exchanges to take place. If the authentication is successful, the Node-F enters the Authenticated state. In this stage there is bi-directional communication between the Node-F and the Node-G. If the authentication is not successful, the Node-F enters Disconnected state. Once in Authenticated state, the Node-F can downgrade its connectivity to Semi-Associated mode.

The management of the node in Authenticated state is performed with COOL [I-D.veillette-core-cool]. As an example, managing the parameters of a Semtech LoRa device can be achieved through the use of the YANG module defined in [I-D.pelov-yang-lora]

Finally, the Node-F may decide to dissociate from the network by sending an explicit request. Upon dissociation the Node-G may release all contexts related to the Node-F and re-association

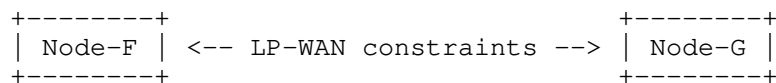
requires going through the authentication stage again. Node mobility is achieved by explicitly dissociating from the old Node-G and then authenticating to the new Node-G. Implicit dissociation is also possible upon the expiration of predefined timers, or in case of mobility optimization.

3.3. CoAP as Signaling Protocol for LP-WANs

Use as CoAP for signaling is implemented as follows. The MAC, network and/or transport layers MUST provide a mechanism to differentiate user data from signaling data frames (e.g. by using separate MAC addresses, IP addresses and/or UDP-ports). Both the Node-G and the Node-F are running CoAP servers for implementing the control plane. Frames exchanged over the LP-WAN radio interface and marked as "signaling data" are handled by the corresponding control plane CoAP servers.

The Node-G runs a (virtual) CoAP server for each Node-F. This server is identified with a DNS name, e.g. "node123.home.node-g.example.com", which can be used explicitly in the CoAP messages via the Proxy-Uri option if needed.

Note, that the Node-R acts only as a transceiver and as such is transparent from protocol point of view. As such, the following management scheme applies:



Node-F connectivity from protocol point of view.

Figure 4

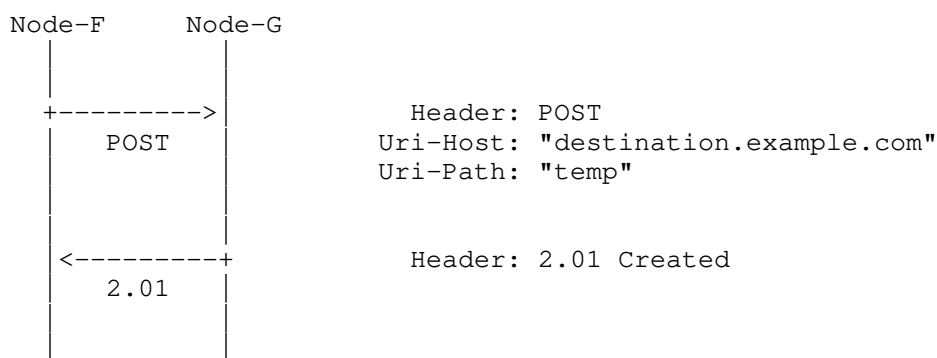
3.3.1. Semi-Association

When in a semi-associated state, a Node-F broadcasts its messages without performing network discovery, or association. If the Node-F is under the coverage of a Node-G, the Node-G will receive the broadcast, and forward the user data. The frames SHOULD be signed, so that they could be authenticated by the network. Layer 2 acknowledgements MUST be used, and in some cases piggybacking on them can provoke the Node-F to associate to the network.

The broadcast messages MUST include the necessary information to join the user data destination, and enough information for the Node-G to authenticate the message sender. This can be achieved through a Confirmable CoAP message, where the user data are POSTed to a well-

known resource defined on the Node-G. DTLS with integrity check can be used, with long-lived keys negotiated by the Node-F and the network. Alternatively, COSE objects may provide the necessary mechanisms.

Even though an application can be implemented by using only simplex association capabilities, there are non-negligible negative consequences related to scalability and evolvability in this case. For example, a Node-F which periodically broadcasts information will occupy the spectrum, even if there is no operator willing to accept its traffic. In addition, no channel access management can be applied.



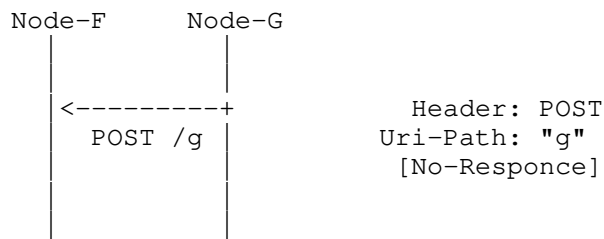
Sending a message in a semi-associated state.

Figure 5

3.3.2. Network Discovery

A network can be discovered by a Node-F reactively or proactively.

Reactive network discovery is based on the detection of periodic beacons emitted by the Node-G. The beacons are implemented with CoAP messages with the No-Response option [I-D.tcs-coap-no-response-option]. The Node-G POSTs its information to a well-known resource, e.g. "/network/node-G/" or a resource alias "/g". Alternatively, this could be achieved by POST-ting to a COOL container (e.g. POST /cool with data node ID = 1 for example).

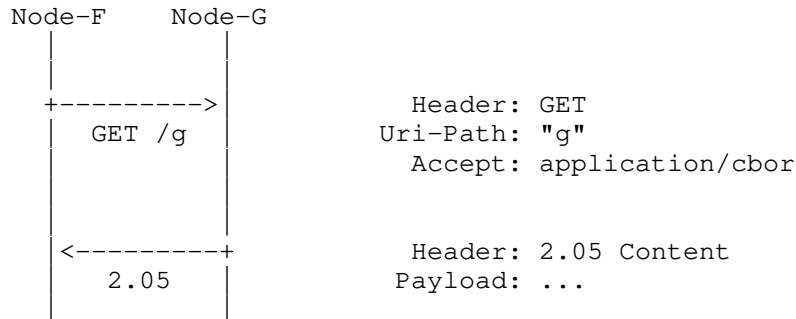


Reactive network discovery. The Node-G sends periodically beacon messages, containing information pertinent to this network.

Figure 6

The CoAP POST request is processed at the Node-F. A resource is created locally, with the representation, which provides the appropriate network parameters, e.g. network ID, Node-G ID, and other radio-related parameters, such as channel, beacon frequency and so forth. This information allows the Node-F to begin the authentication phase.

A Node-F may chose to proactively probe for the existence of network coverage. In that case, it sends a Confirmable CoAP GET request to obtain the information from a well-known resource, normally published by the beacon messages, e.g. "/network/node-G/" or a resource alias "/g" or COOL data node ID ("/cool" data node ID = 2 for example).



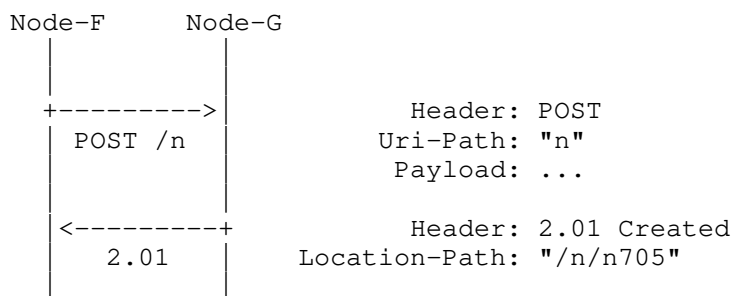
Proactive network discovery. The Node-F request the information of all surrounding Node-Gs.

Figure 7

Once the network is discovered, the Node-F has all necessary information to start the authentication phase.

3.3.3. Association

Before being able to communicate, the Node-F must associate to the network, and then eventually authenticate. The association phase signals to the Node-G that there is a new device willing to communicate with the network. This association SHOULD provide enough information to allow the Node-G to start the authentication process. For example, it may provide the AAA server, which could authenticate the Node-F, or its EAP-Identity. Note, that the Node-F may elect to mark the association message with the No-response option [I-D.tcs-coap-no-response-option], waiting for the subsequent authentication request from the Node-G.



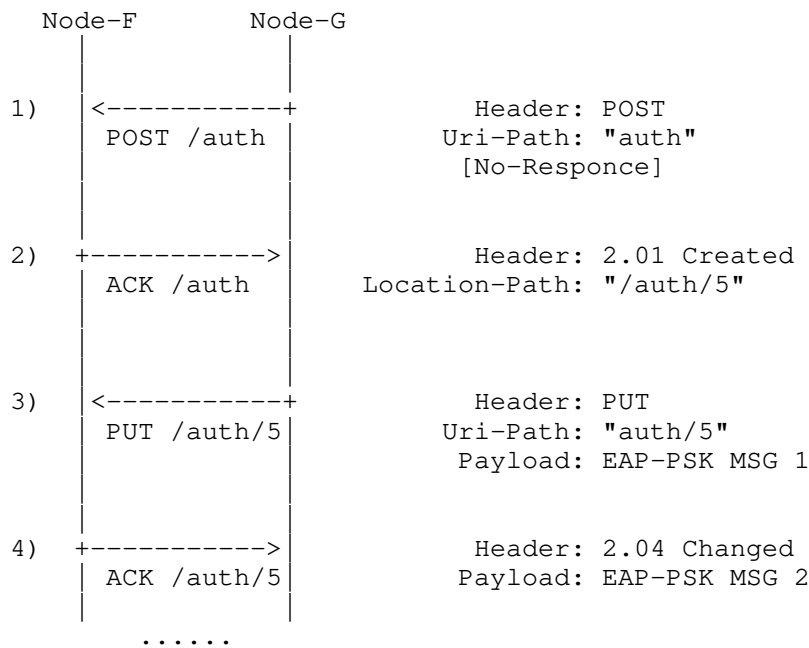
Node-F associates to a network, by creating a corresponding resource element on the Node-G.

Figure 8

3.3.4. Authentication

The EAP-over-CoAP [I-D.marin-ace-wg-coap-eap] specifies an approach to encapsulating EAP messages over CoAP. This allows to authenticate a Node-F, which wishes to join an LP-WAN, and negotiate the L2 encryption keys, and DTLS keying material.

As the Node-F has already associated to the Node-G, it is the Node-G that initiates the authentication request, by going directly to Step 1) of the EAP-over-CoAP specification.



Node-F and Node-G perform mutual authentication following EAP-over-CoAP.

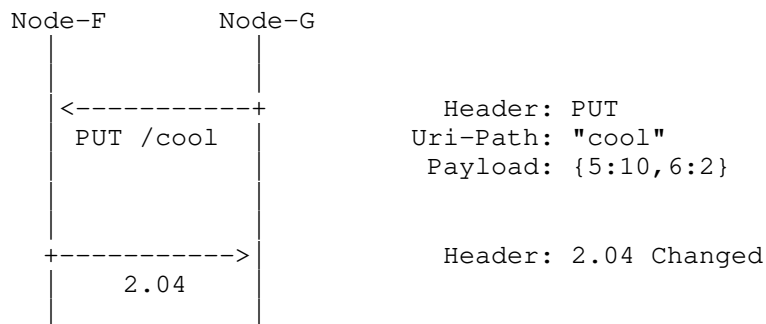
Figure 9

Upon the end of the authentication phase, a Master Shared Key (MSK) is known by the Node-F and the Node-G, and is used to generate DTLS encryption or integrity keys. Further communications should be encrypted/signed with the freshly derived keys.

3.3.5. Operation

Once the Node-F is authenticated to the network, it can send user data via the Node-G to any other end-point on the Internet.

During the operation of the Node-F, the network may need to change one or more parameters concerning the LP-WAN radio parameters of the Node-F. These changes may even concern parameters related to the Node-F itself (such as sleep cycles), its network parameters (e.g. IP addresses), and so forth. This is achieved through the use of COOL [I-D.veillette-core-cool]. The appropriate YANG modules must be present on the Node-F (e.g. a Semtech LoRa Node-F should implement the [I-D.pelov-yang-lora] module).

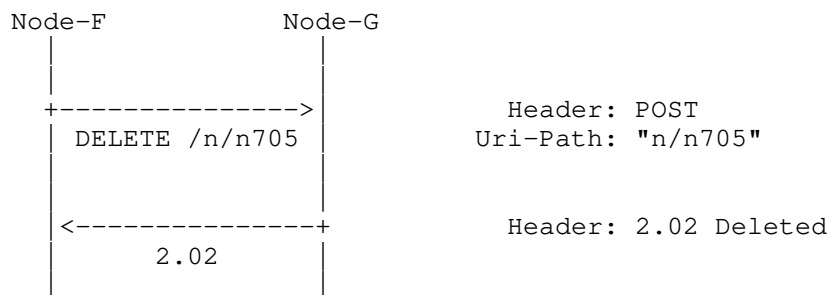


Example, in which the Node-G changes the Spreading Factor (e.g. COOL data node ID = 5) to 10, and the Channel (e.g. COOL data node ID = 6) to 2. Note, that the payload is encoded in CBOR.

Figure 10

3.3.6. Dissociation

If the Node-F wishes to deregister from the network, it could do so by deleting the context created upon association:



Node-F dissociates from the network by deleting its associated resources.

Figure 11

4. Acknowledgements

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.

7. References

7.1. Normative References

- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-16 (work in progress), December 2014.
- [I-D.marin-ace-wg-coap-eap]
Garcia, D., "EAP-based Authentication Service for CoAP", draft-marin-ace-wg-coap-eap-01 (work in progress), October 2014.
- [I-D.pelov-yang-lora]
Pelov, A., Toutain, L., Delibie, Y., and A. Minaburo, "YANG module for LoRa Networks", draft-pelov-yang-lora-00 (work in progress), December 2015.
- [I-D.tcs-coap-no-response-option]
Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "CoAP option for no server-response", draft-tcs-coap-no-response-option-13 (work in progress), November 2015.
- [I-D.veillette-core-cool]
Veillette, M. and A. Pelov, "Constrained Objects Language", draft-veillette-core-cool-00 (work in progress), November 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

7.2. Informative References

- [IEEE.802-15.4k]
Institute of Electrical and Electronics Engineers, "Low-Rate Wireless Personal Area Networks (LR-WPANs) - Amendment 5: Physical Layer Specifications for Low Energy, Critical Infrastructure Monitoring Networks., IEEE 802.15.4k", IEEE Standard 802.15.4, 2013.
- [LoRa]
Semtech, "<https://web.archive.org/web/20150510011904/https://www.semtech.com/wireless-rf/lora.html>", May 2015.
- [LTN001]
European Telecommunications Standards Institute, "Low Throughput Networks (LTN); Use Cases for Low Throughput Networks, ETSI GS LTN 001", IEEE ETSI GS LTN 001, 2014.
- [LTN003]
European Telecommunications Standards Institute, "Low Throughput Networks (LTN); Protocols and Interfaces, ETSI GS LTN 003", IEEE ETSI GS LTN 003, 2014.
- [RFC3552]
Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<http://www.rfc-editor.org/info/rfc3552>>.
- [SigFox]
SigFox, "<https://web.archive.org/web/20150628225901/http://www.sigfox.com/en/#!/technology>", June 2015.

Authors' Addresses

Alexander Pelov (editor)
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
FR

Email: a@ackl.io

Laurent Toutain (editor)
Institut MINES-TELECOM ; TELECOM Bretagne
2 rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
FR

Email: laurent.toutain@telecom-bretagne.eu

Yannick Delibie (editor)
Kerlink
1 rue Jacqueline Auriol
Thorigne-Fouillard, Bretagne 35235
FR

Email: yannick.delibie@kerlink.fr

CORE WG
Internet-Draft
Intended status: Informational
Expires: September 19, 2016

A. Rahman
C. Wang
InterDigital Communications, LLC
March 18, 2016

Advanced Resource Directory Features
draft-rahman-core-advanced-rd-features-02

Abstract

The Resource Directory (RD) is a key element for successful deployments of constrained networks. Similar to the HTTP web search engines (e.g. Google, Bing), the RD for CoAP should also support useful search query responses beyond a basic listing of relevant links. This document proposes several new features to be considered for the RD. The only goal of this document is to trigger discussion in the CoRE WG so that all relevant features for RD evolution are taken into account during CoRE re-charter activities.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 19, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminology and Conventions	2
2. Background	2
3. Proposal	3
4. Summary	4
5. Acknowledgements	4
6. IANA Considerations	4
7. Security Considerations	4
8. References	4
8.1. Normative References	4
8.2. Informative References	5
Authors' Addresses	5

1. Terminology and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document assumes readers are familiar with the terms and concepts that are used in [RFC6690], [RFC7252] and [I-D.ietf-core-resource-directory].

2. Background

The concept of the Resource Directory (RD) is described in [I-D.ietf-core-resource-directory]. It is defined as a node which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources. The [I-D.ietf-core-resource-directory] specifies the web interfaces that a Resource Directory supports in order for devices to discover the RD and to register, maintain, lookup and remove resources descriptions.

The relevant specification of interfaces in [I-D.ietf-core-resource-directory] is given using the CoAP protocol [RFC7252] for all interfaces. Also, HTTP protocol [RFC7230] support is given for some interfaces. For example, all the response codes (i.e. success and error) for registering and looking up resources support both CoAP and HTTP. However, the important multicast discovery interface does not support HTTP. The Group interface also does not support HTTP.

The CoRE Link Format [RFC6690] describes the format of the payload of a CoAP message that carries a set of CoAP URIs. With relation to the RD, the CoRE Link Format is to be used by a device to carry (encode) the set of URIs it wants to register with an RD. Also, the CoRE Link Format is used to carry (encode) the set of URIs returned by a RD for a lookup query (including the initial multicast discovery request). While in theory the CoRE Link Format [RFC6690] specification states that it may be used with HTTP, in practice many details still need to be fleshed out and specified before this can be realized.

3. Proposal

It is proposed that the RD should also support the following additional features:

1. Explicit HTTP Support - Though there is some support of HTTP in [I-D.ietf-core-resource-directory], the specification should be further expanded to also explicitly support HTTP for the Discovery and perhaps the Group Functions. Also, the RD function is intimately tied to the CoRE Link Format [RFC6690] which does not have any explicit support of HTTP at all. So the CoRE Link Format definitely needs to be updated to support HTTP explicitly.
2. Mirror Server - The CoRE WG has previously discussed the concept of a mirror server in relation to supporting sleepy devices. Specifically, [I-D.vial-core-mirror-server] recommends to create a new class of RDs which store the actual resource representations (as opposed to simply storing the URI) in a special type of RD called the Mirror Server. Communicating devices can both lookup the resource, and then also fetch directly the resource representation, from the Mirror Server regardless of the state of the sleepy server.
3. Re-direction to another RD - A given RD may not have the URIs being queried for registered in its database. The given RD should have the capability to re-direct the querying client to another RD which may have the information of interest.
4. URI Ranking - Current Internet search engines (e.g. Google) have extensive methods for ranking the URIs returned to a human initiated search query. For example, the concept of Search Engine Optimization (SEO) has spawned a large industry in the web world for specifically this purpose. The concept of URI ranking (to indicate the "value" of the URI) should also be supported by the RD.
5. Indication of transport protocol - Several proposals exist (e.g. [I-D.silverajan-core-coap-alternative-transports]) in the CoRE WG to support alternative transports (e.g. TCP, SMS) for CoAP beyond the

current UDP transport. It would be very useful if search results from a RD indicated the type of transport supported by a given URI.

6. Privacy Model - IoT devices may often contain sensitive information (e.g. health monitoring device) or affect human safety (e.g. traffic light controllers, elevator actuators). When the resources of a device is registered with a given RD and domain, should anyone at all be able to easily discover the resources associated with the device? Does this cause privacy or security concerns in certain RD lookup scenarios? Currently, [I-D.ietf-core-resource-directory] has a very brief mention that endpoint and clients should be authenticated and access controlled. However, a more complete privacy model should be developed to address this very important issue.

4. Summary

The proposed set of feature extensions for the RD will improve the constrained environment search capability and make deployments more efficient. These RD feature extensions should be individually considered during the CoRE re-charter discussions. Evolution and forward thinking is required for the CoRE RD, as constantly occurs in the current Internet for HTTP web search engines (e.g. Google).

5. Acknowledgements

TBD.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

Not applicable.

8. References

8.1. Normative References

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-05 (work in progress), October 2015.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

8.2. Informative References

- [I-D.silverajan-core-coap-alternative-transport] Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transport-09 (work in progress), December 2015.
- [I-D.vial-core-mirror-server] Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-server-01 (work in progress), April 2013.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<http://www.rfc-editor.org/info/rfc7390>>.

Authors' Addresses

Akbar Rahman
InterDigital Communications, LLC

Email: akbar.rahman@interdigital.com

Chonggang Wang
InterDigital Communications, LLC

Email: chonggang.wang@interdigital.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 20, 2016

T. Savolainen
Nokia
K. Hartke
Universitaet Bremen TZI
B. Silverajan
Tampere University of Technology
March 19, 2016

CoAP over WebSockets
draft-savolainen-core-coap-websockets-06

Abstract

This document specifies how to retrieve and update CoAP resources using CoAP requests and responses over the WebSocket Protocol.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 20, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Overview	3
1.2. Terminology	5
2. CoAP over WebSockets	5
2.1. Opening Handshake	5
2.2. Message Format	6
2.3. Message Transmission	7
2.4. Connection Health	7
2.5. Closing the Connection	8
3. CoAP over WebSockets URIs	8
4. Security Considerations	9
5. IANA Considerations	9
5.1. URI Scheme Registrations	9
5.2. WebSocket Subprotocol Registration	11
5.3. Well-Known URI Suffix Registration	11
6. References	12
6.1. Normative References	12
6.2. Informative References	12
Appendix A. Examples	13
Acknowledgements	16
Authors' Addresses	16

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web protocol designed for communications between resource constrained nodes. By default, CoAP operates on top of UDP or DTLS, but there is interest in using CoAP also over other types of transports, such as SMS [I-D.becker-core-coap-sms-gprs].

An interesting transport for CoAP could be the WebSocket Protocol [RFC6455]. The WebSocket protocol provides two-way communication between a client and a server after upgrading an HTTP [RFC7230] connection, and may be available in an environment that does not allow transportation of CoAP over UDP. This environment can be, for example, a corporate network with Internet access only via an HTTP proxy, or a CoAP application running inside a web browser without access to connectivity means other than HTTP and WebSockets.

This document specifies how to access resources using CoAP requests and responses over the WebSocket Protocol. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either by communicating CoAP directly with a CoAP server that is accessible over a WebSocket Connection, or via an intermediary that proxies CoAP requests and responses between different transports, such as between WebSockets and UDP.

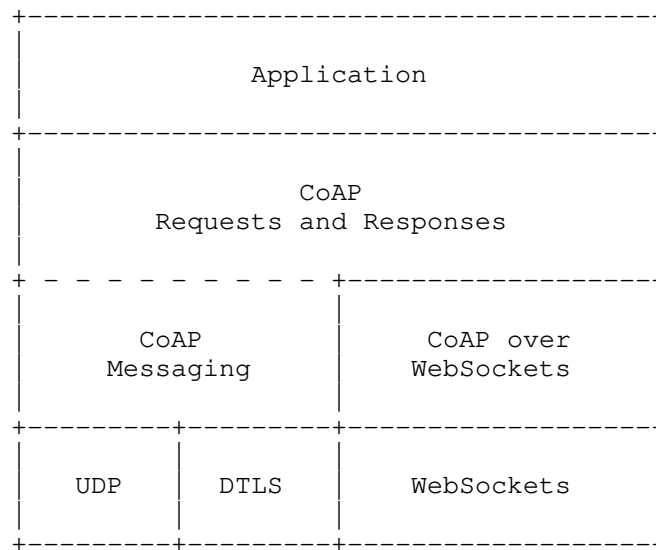


Figure 1: Abstract layering of CoAP extended by WebSockets

1.1. Overview

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client seeking to retrieve or update a CoAP resource located at a CoAP server that exposes a WebSocket endpoint (Figure 2). The CoAP client takes the role of the WebSocket client, establishes a WebSocket Connection and sends a CoAP request, to which the CoAP server returns a CoAP response. The WebSocket Connection can be used for any number of requests.

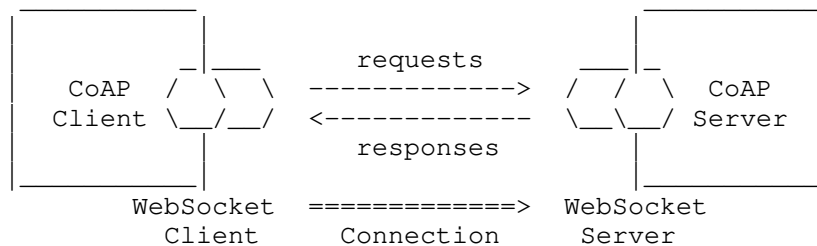


Figure 2: CoAP client (WebSocket client) accesses CoAP server (WebSocket server)

The challenge in this configuration is to identify resource in the namespace of the CoAP server: When the WebSocket Protocol is used by a dedicated client directly (i.e., not from a web page through a web browser), the client can connect to any WebSocket endpoint. This means it is necessary that the client is able to determine both the WebSocket endpoint (identified by a "ws" or "wss" URI) and the path and query of the CoAP resource within that endpoint from the same URI. When the WebSocket Protocol is used from a web page, the choices are more limited [RFC6454], but the challenge persists.

Section 3 proposes a new "coap+ws" URI scheme that identifies both a WebSocket endpoint and a resource within that endpoint as follows:

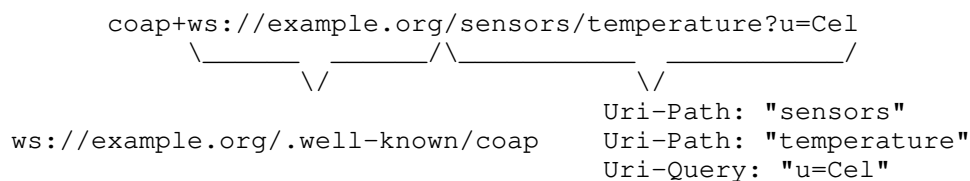


Figure 3: The "coap+ws" URI Scheme

Another possible configuration is to set up a CoAP forward proxy at the WebSocket endpoint. Depending on what transports are available to the proxy, it could forward the request to a CoAP server with a CoAP UDP endpoint (Figure 4), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The client specifies the resource to be updated or retrieved in the Proxy-URI Option.

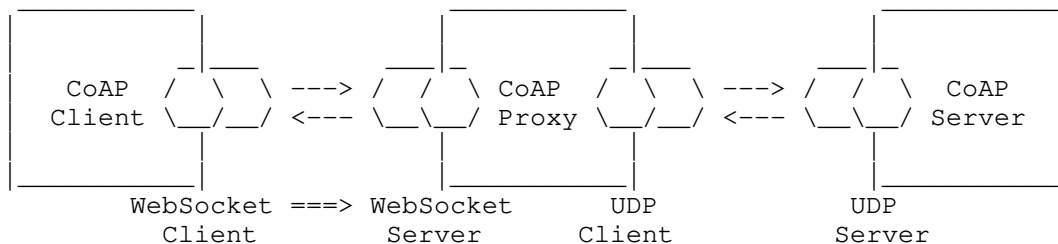


Figure 4: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

A third possible configuration is a CoAP server running inside a web browser (Figure 5). The web browser initially connects to a

WebSocket endpoint and is then reachable through the WebSocket server. When no connection exists, the CoAP server is not reachable; it therefore can be considered a Sleepy Endpoint (SEP) [I-D.dijk-core-sleepy-reqs]. Because the WebSocket server is the only way to reach the CoAP server, the CoAP proxy should be a Reverse Proxy.

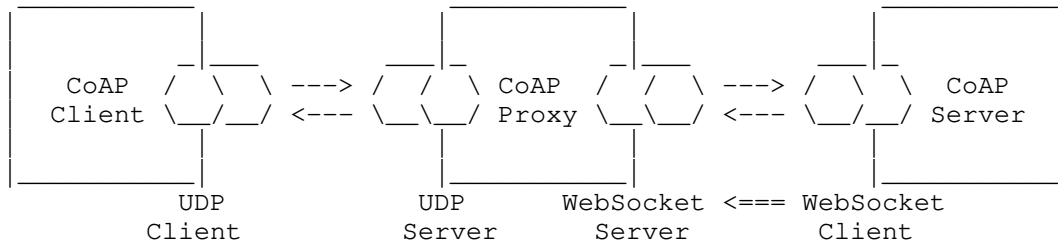


Figure 5: CoAP Client (UDP client) accesses sleepy CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

Further configurations are possible, including those where a WebSocket Connection is established through an HTTP proxy.

1.2. Terminology

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455] and [RFC7252].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. CoAP over WebSockets

CoAP over WebSockets is intentionally very similar to CoAP as defined over UDP. Therefore, instead of presenting CoAP over WebSockets as a new protocol, this document specifies it as a series of deltas from [RFC7252].

2.1. Opening Handshake

Before CoAP requests and responses can be exchanged, a WebSocket Connection needs to be established as defined in Section 4 of [RFC6455]. The WebSocket client MUST include the subprotocol name "coap.v1" in the list of protocols, which indicates support for the protocol defined in this document. Figure 6 shows an example.

```

GET /.well-known/coap HTTP/1.1
Host: example.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: coap.v1
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: coap.v1

```

Figure 6: Example of an Opening Handshake

2.2. Message Format

Once a WebSocket Connection has been established, CoAP requests and responses can be exchanged as WebSocket messages. Since CoAP uses a binary message format, the messages are transmitted in binary data frames as specified in Sections 5 and 6 of [RFC6455].

The message format is very similar to the format specified for CoAP over UDP [RFC7252]. The differences are as follows:

- o Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. This means the "T" and "Message ID" fields in the CoAP message header can be elided.
- o Furthermore, since the CoAP version is already negotiated during the opening handshake, the "Ver" field can be elided as well.

```

      0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  R   | TKL |      Code      |      Token (TKL bytes) ...   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Options (if any) ... |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 1 1 1 1 1 1 |      Payload (if any) ...                 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Figure 7: CoAP Message Format over WebSockets

The resulting message format is shown in Figure 7. The four most-significant bits of the first byte are reserved (R) and MUST be set to zero. The remaining fields and structure are the same as defined in [RFC7252].

Requests and response messages can be fragmented as specified in Section 5.4 of [RFC6455], though typically they are sent unfragmented as they tend to be small and fully buffered before transmission. The WebSocket protocol does not provide means for multiplexing; if it is not desirable for a large message to monopolize the connection, requests and responses can be transferred in a blockwise fashion as defined in [I-D.ietf-core-block].

Messages MUST NOT be Empty (Code 0.00), i.e., messages always carry either a request or a response.

2.3. Message Transmission

CoAP requests and responses are exchanged asynchronously over the WebSocket Connection, i.e., a CoAP client can send multiple requests without waiting for a response, and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Concurrent requests are differentiated by the Token, which is scoped locally to the connection.

The connection is bi-directional, so requests can be sent both by the entity that established the connection and the remote host.

Retransmission and deduplication of messages is provided by the WebSocket Protocol. CoAP over WebSockets therefore does not make a distinction between Confirmable or Non-Confirmable messages, and does not provide Acknowledgement or Reset messages.

Since the WebSocket Protocol provides ordered delivery of messages, the mechanism for reordering detection when observing resources [RFC7641] is not needed. The value of the Observe Option in notifications therefore MAY be empty on transmission and MUST be ignored on reception.

2.4. Connection Health

When a client does not receive any response for some time after sending a CoAP request (or, similarly, when a client observes a resource and it does not receive any notification for some time), the connection between the WebSocket client and the WebSocket server may be lost or temporarily disrupted without the client being aware of it.

To check the health of the WebSocket Connection (and thereby of all active requests, if any), the client can send a Ping frame or an unsolicited Pong frame as specified in Section 5.5 of [RFC6455].

2.5. Closing the Connection

The WebSocket Connection is closed as specified in Section 7 of [RFC6455].

All requests for which the CoAP client has not received a response yet, are cancelled when the connection is closed. If the client observes one or more resource over the WebSocket Connection, then the CoAP server (or intermediary in the role of the CoAP server) MUST remove all entries associated with the client from the lists of observers when the connection is closed.

3. CoAP over WebSockets URIs

For the first configuration discussed in Section 1.1, this document defines two new URIs schemes that can be used for identifying CoAP resources and providing a means of locating these resources: "coap+ws" and "coap+wss".

Similar to the "coap" and "coaps" schemes, the "coap+ws" and "coap+wss" schemes organize resources hierarchically under a CoAP origin server. The key difference is that the server is potentially reachable on a WebSocket endpoint instead of a UDP endpoint.

The WebSocket endpoint is identified by a "ws" or "wss" URI that is composed of the authority part of the "coap+ws" or "coap+wss" URI, respectively, and the well-known path "/.well-known/coap" [RFC5785]. The path and query parts of a "coap+ws" or "coap+wss" URI identify a resource within the specified endpoint which can be operated on by the methods defined by the CoAP protocol.

The syntax of the "coap+ws" and "coap+wss" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty" and "query" are the same as in [RFC3986].

```
coap-ws-URI =  
    "coap+ws:" "/" host [ ":" port ] path-abempty [ "?" query ]  
  
coap-wss-URI =  
    "coap+wss:" "/" host [ ":" port ] path-abempty [ "?" query ]
```

The port component is OPTIONAL; the default for "coap+ws" is port 80, while the default for "coap+wss" is port 443.

Fragment identifiers are not part of the request URI and thus MUST NOT be transmitted in a WebSocket handshake or in the URI options of a CoAP request.

4. Security Considerations

CoAP over WebSockets and CoAP over TLS-secured WebSockets do not introduce additional security issues beyond CoAP and DTLS-secured CoAP respectively [RFC7252].

The security considerations of [RFC6455] apply.

5. IANA Considerations

[Note to RFC Editor: Please replace XXXX in this section with the RFC number of this specification.]

5.1. URI Scheme Registrations

5.1.1. "coap+ws"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+ws".

URI scheme name.
coap+ws

Status.
Permanent.

URI scheme syntax.
Defined in Section 3 of [RFCXXXX].

URI scheme semantics.
The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol.

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Interoperability considerations.
None.

Security considerations.
See Section 4 of [RFCXXXX].

Contact.
IETF Chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
[RFCXXXX]

5.1.2. "coap+wss"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+wss".

URI scheme name.
coap+wss

Status.
Permanent.

URI scheme syntax.
Defined in Section 3 of [RFCXXXX].

URI scheme semantics.
The "coap+wss" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol secured with Transport Layer Security (TLS).

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Interoperability considerations.

None.

Security considerations.

See Section 4 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

5.2. WebSocket Subprotocol Registration

This document requests the registration of the subprotocol name "coap.v1" in the WebSocket Subprotocol Name Registry.

Subprotocol Identifier.

coap.v1

Subprotocol Common Name.

Constrained Application Protocol (CoAP)

Subprotocol Definition.

[RFCXXXX]

5.3. Well-Known URI Suffix Registration

This document requests the registration of the Well-Known URI suffix "coap" in the Well-Known URI Registry.

URI suffix.

coap

Change controller.

IETF

Specification document(s).

[RFCXXXX]

Related information.

None.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<http://www.rfc-editor.org/info/rfc5785>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<http://www.rfc-editor.org/info/rfc6455>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

6.2. Informative References

- [I-D.becker-core-coap-sms-gprs]
Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-05 (work in progress), August 2014.
- [I-D.dijk-core-sleepy-reqs]
Dijk, E., "Sleepy Devices using CoAP - Requirements", draft-dijk-core-sleepy-reqs-00 (work in progress), June 2013.

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<http://www.rfc-editor.org/info/rfc6454>>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.

Appendix A. Examples

This section gives examples for the first two configurations discussed in Section 1.1.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be as follows. Figure 8 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap+ws://example.org/sensors/temperature?u=Cel>, for example, from a resource representation that it retrieved previously.
2. It establishes a WebSocket Connection to the endpoint URI composed of the authority "example.org" and the well-known path "/.well-known/coap", <ws://example.org/.well-known/coap>.
3. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.
4. It waits for the server to return a response.
5. The CoAP client uses the connection for further requests, or the connection is closed.

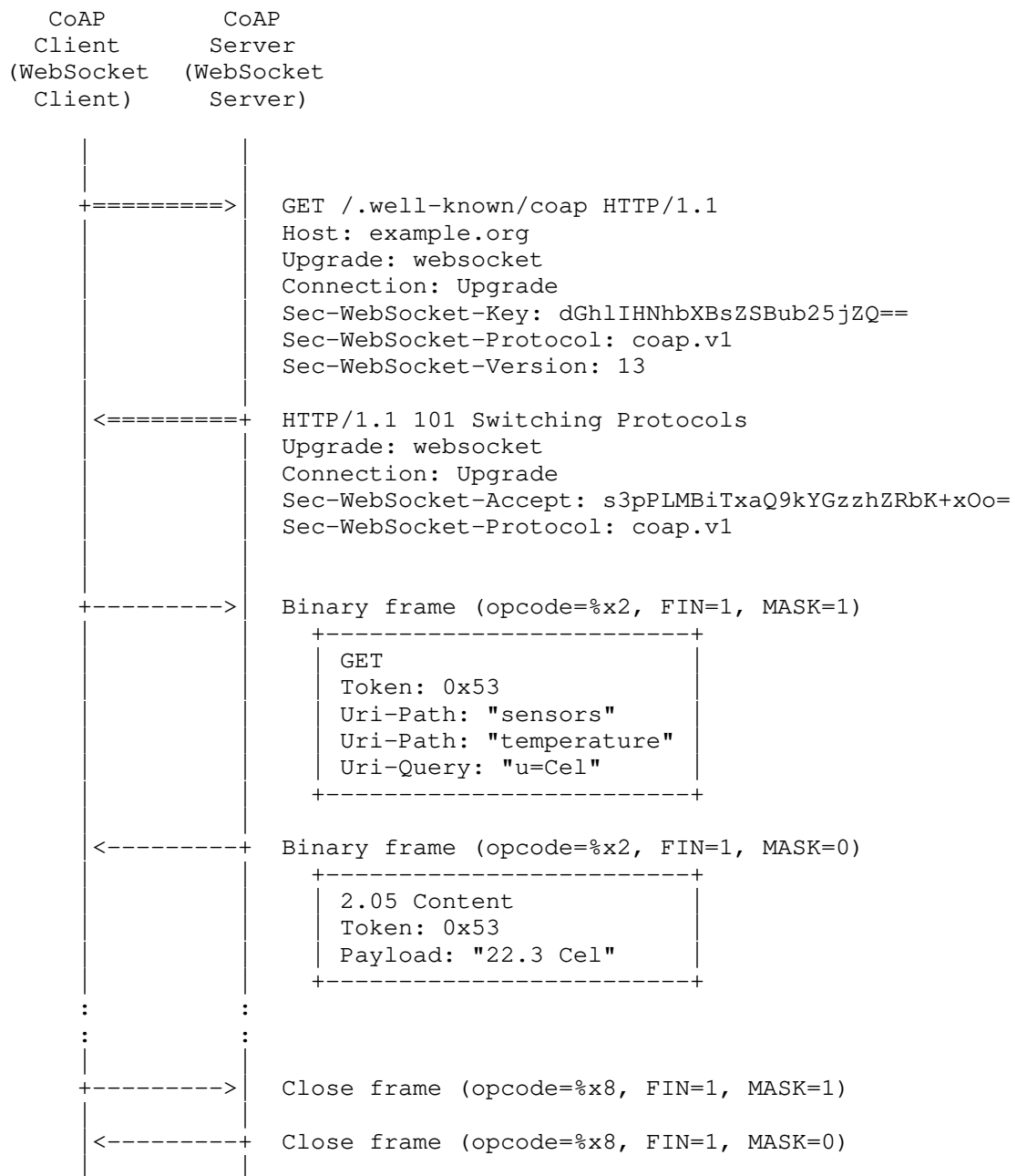


Figure 8: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 9 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource `<coap://[2001:DB8::1]/>`. The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response over the WebSocket connection to the client.

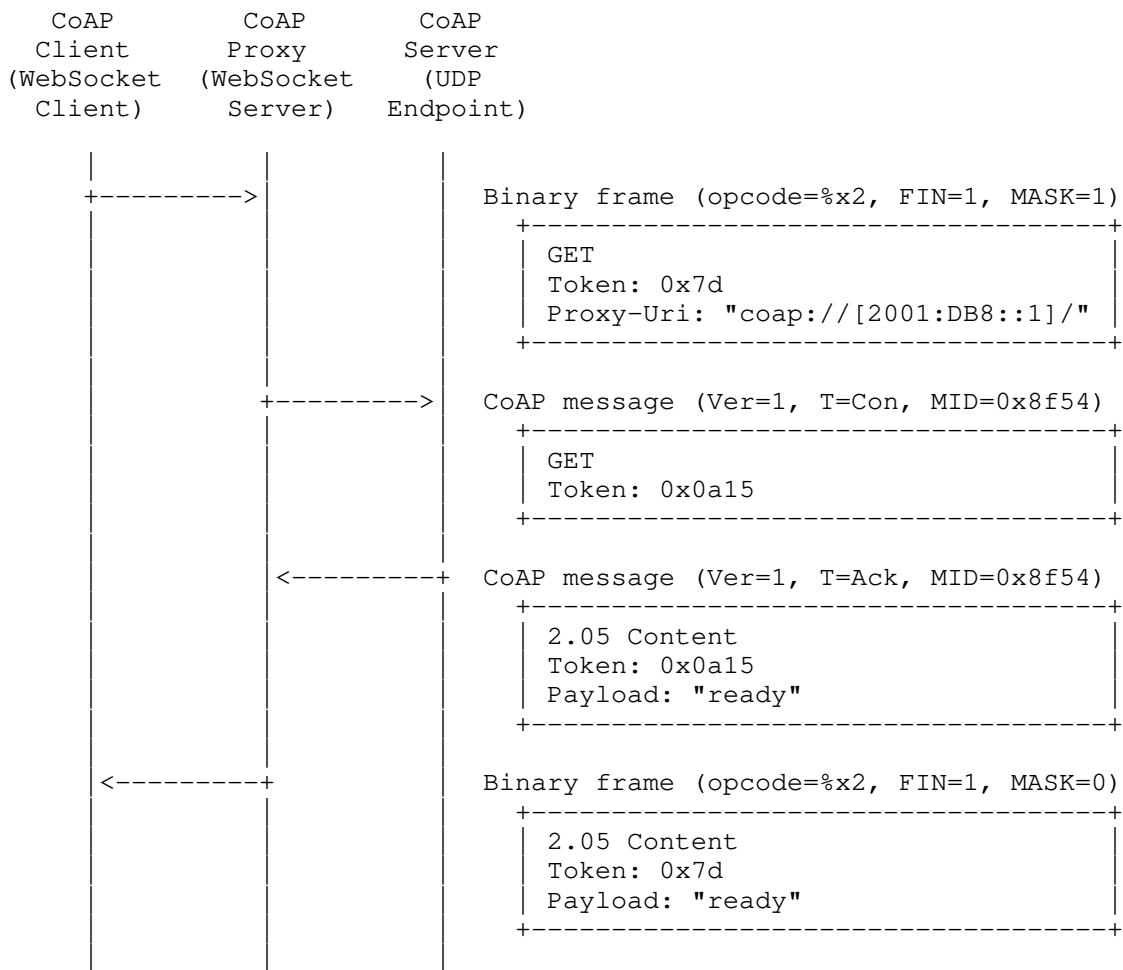


Figure 9: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

Acknowledgements

Thanks to Nadir Javed for helpful comments and discussions that have shaped the document.

Authors' Addresses

Teemu Savolainen
Nokia
Hermiankatu 12 D
Tampere FI-33720
Finland

Email: teemu.savolainen@nokia.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 18, 2016

T. Savolainen
Nokia
K. Hartke
Universitaet Bremen TZI
B. Silverajan
Tampere University of Technology
June 16, 2016

CoAP over WebSockets
draft-savolainen-core-coap-websockets-07

Abstract

This document specifies how to retrieve and update CoAP resources using CoAP requests and responses over the WebSocket Protocol.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 18, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Overview	3
1.2. Terminology	5
2. CoAP over WebSockets	5
2.1. Opening Handshake	5
2.2. Message Format	6
2.3. Message Transmission	7
2.4. Connection Health	7
2.5. Closing the Connection	8
3. CoAP over WebSockets URIs	8
3.1. Decomposing and Composing URIs	9
4. Security Considerations	9
5. IANA Considerations	9
5.1. URI Scheme Registrations	9
5.2. WebSocket Subprotocol Registration	11
5.3. Well-Known URI Suffix Registration	12
6. References	12
6.1. Normative References	12
6.2. Informative References	13
Appendix A. Examples	13
Acknowledgements	17
Authors' Addresses	17

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web protocol designed for communications between resource constrained nodes. By default, CoAP operates as a layer on top of UDP or DTLS, but there is interest in using CoAP also over other types of transports, such as TCP, TLS [I-D.ietf-core-coap-tcp-tls], or SMS [I-D.becker-core-coap-sms-gprs].

An interesting transport for CoAP could be the WebSocket Protocol [RFC6455]. The WebSocket protocol provides two-way communication between a client and a server after upgrading an HTTP/1.1 [RFC7230] connection, and may be available in an environment that does not allow transportation of CoAP over UDP. This environment can be, for example, a corporate network with Internet access only via an HTTP proxy, or a CoAP application running inside a web browser without access to connectivity means other than HTTP and WebSockets.

This document specifies how to access resources using CoAP requests and responses over the WebSocket Protocol. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either by communicating CoAP directly with a CoAP server accessible over a WebSocket Connection or via a CoAP intermediary that proxies CoAP

requests and responses between different transports, such as between WebSockets and UDP.

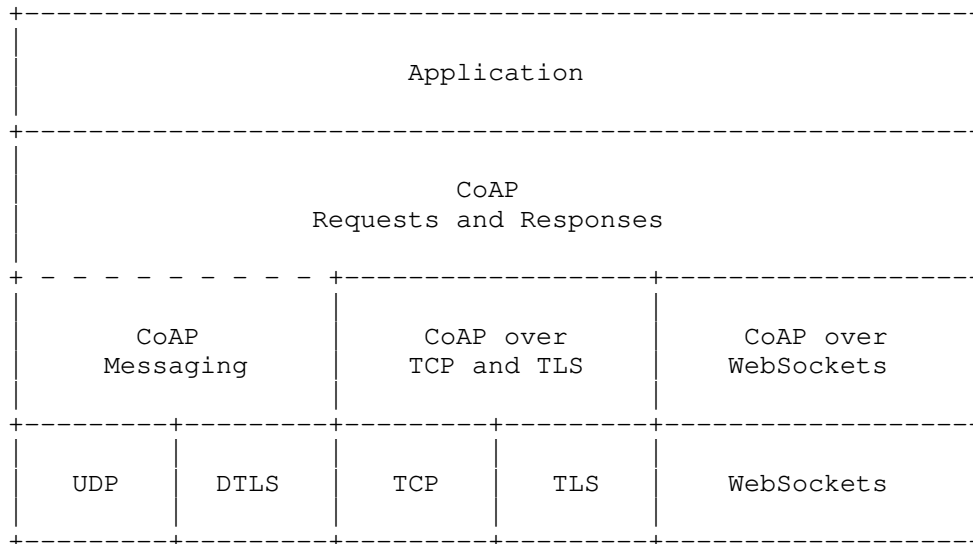


Figure 1: Abstract layering of CoAP extended by WebSockets

1.1. Overview

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client seeking to retrieve or update a CoAP resource located at a CoAP server that exposes a WebSocket endpoint (Figure 2). The CoAP client takes the role of the WebSocket client, establishes a WebSocket Connection and sends a CoAP request, to which the CoAP server returns a CoAP response. The WebSocket Connection can be used for any number of requests.

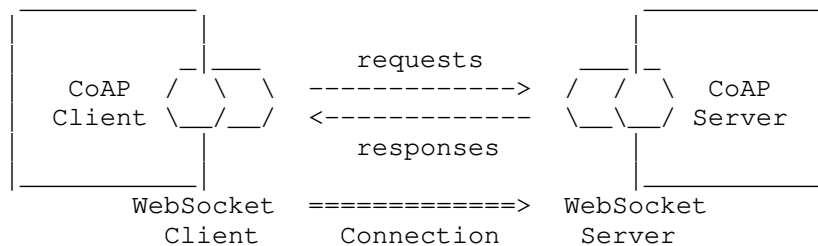


Figure 2: CoAP client (WebSocket client) accesses CoAP server (WebSocket server)

The challenge in this configuration is to identify resource in the namespace of the CoAP server: When the WebSocket Protocol is used by a dedicated client directly (i.e., not from a web page through a web browser), the client can connect to any WebSocket endpoint. This means it is necessary that the client is able to determine both the WebSocket endpoint (identified by a "ws" or "wss" URI) and the path and query of the CoAP resource within that endpoint from the same URI. When the WebSocket Protocol is used from a web page, the choices are more limited [RFC6454], but the challenge persists.

Section 3 proposes a new "coap+ws" URI scheme that identifies both a WebSocket endpoint and a resource within that endpoint as follows:

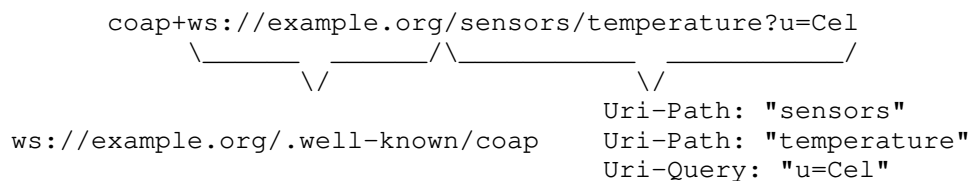


Figure 3: The "coap+ws" URI Scheme

Another possible configuration is to set up a CoAP forward proxy at the WebSocket endpoint. Depending on what transports are available to the proxy, it could forward the request to a CoAP server with a CoAP UDP endpoint (Figure 4), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The client specifies the resource to be updated or retrieved in the Proxy-URI Option.

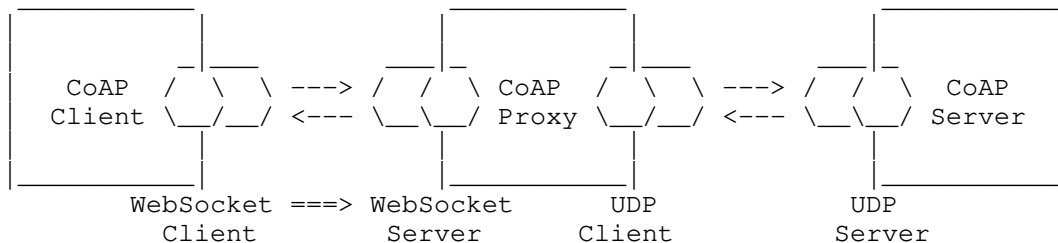


Figure 4: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

A third possible configuration is a CoAP server running inside a web browser (Figure 5). The web browser initially connects to a WebSocket endpoint and is then reachable through the WebSocket server. When no connection exists, the CoAP server is not reachable; it therefore can be considered a Sleepy Endpoint (SEP) [I-D.dijk-core-sleepy-reqs]. Because the WebSocket server is the

only way to reach the CoAP server, the CoAP proxy should be a Reverse Proxy.

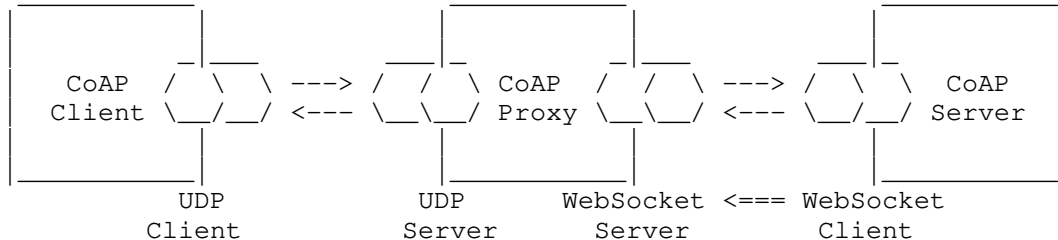


Figure 5: CoAP Client (UDP client) accesses sleepy CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

Further configurations are possible, including those where a WebSocket Connection is established through an HTTP proxy.

1.2. Terminology

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455] and [RFC7252].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. CoAP over WebSockets

CoAP over WebSockets is intentionally very similar to CoAP as defined over UDP. Therefore, instead of presenting CoAP over WebSockets as a new protocol, this document specifies it as a series of deltas from [RFC7252].

2.1. Opening Handshake

Before CoAP requests and responses can be exchanged, a WebSocket Connection needs to be established as defined in Section 4 of [RFC6455]. Figure 6 shows an example.

The WebSocket client **MUST** include the subprotocol name "coap" in the list of protocols, which indicates support for the protocol defined in this document. Any later, incompatible versions of CoAP or CoAP over WebSockets will use a different subprotocol name.

The WebSocket client includes the hostname of the WebSocket server in the Host header field of its handshake as per [RFC6455]. The Host

header field also indicates the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server.

```
GET /.well-known/coap HTTP/1.1
Host: example.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: coap
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: coap
```

Figure 6: Example of an Opening Handshake

2.2. Message Format

Once a WebSocket Connection has been established, CoAP requests and responses can be exchanged as WebSocket messages. Since CoAP uses a binary message format, the messages are transmitted in binary data frames as specified in Sections 5 and 6 of [RFC6455].

The message format is very similar to the format specified for CoAP over UDP [RFC7252]. The differences are as follows:

- o Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. This means the "T" and "Message ID" fields in the CoAP message header can be elided.
- o Furthermore, since the CoAP version is already negotiated during the opening handshake, the "Ver" field can be elided as well.

```

      0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  R   | TKL |      Code      |      Token (TKL bytes) ...   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Options (if any) ... |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 1 1 1 1 1 1 |      Payload (if any) ...   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 7: CoAP Message Format over WebSockets

The resulting message format is shown in Figure 7. The four most-significant bits of the first byte are reserved (R) and MUST be set to zero. The remaining fields and structure are the same as defined in [RFC7252].

Requests and response messages can be fragmented as specified in Section 5.4 of [RFC6455], though typically they are sent unfragmented as they tend to be small and fully buffered before transmission. The WebSocket protocol does not provide means for multiplexing; if it is not desirable for a large message to monopolize the connection, requests and responses can be transferred in a blockwise fashion as defined in [I-D.ietf-core-block].

Messages MUST NOT be Empty (Code 0.00), i.e., messages always carry either a request or a response.

2.3. Message Transmission

CoAP requests and responses are exchanged asynchronously over the WebSocket Connection, i.e., a CoAP client can send multiple requests without waiting for a response and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Concurrent requests are differentiated by their Token, which are scoped locally to the connection.

The connection is bi-directional, so requests can be sent both by the entity that established the connection and the remote host.

Retransmission and deduplication of messages is provided by the WebSocket Protocol. CoAP over WebSockets therefore does not make a distinction between Confirmable or Non-Confirmable messages, and does not provide Acknowledgement or Reset messages.

Since the WebSocket Protocol provides ordered delivery of messages, the mechanism for reordering detection when observing resources [RFC7641] is not needed. The value of the Observe Option in notifications therefore MAY be empty on transmission and MUST be ignored on reception.

2.4. Connection Health

When a client does not receive any response for some time after sending a CoAP request (or, similarly, when a client observes a resource and it does not receive any notification for some time), the connection between the WebSocket client and the WebSocket server may be lost or temporarily disrupted without the client being aware of it.

To check the health of the WebSocket Connection (and thereby of all active requests, if any), the client can send a Ping frame or an unsolicited Pong frame as specified in Section 5.5 of [RFC6455]. There is no way to retransmit a request without creating a new one. Re-registering interest in a resource is permitted, but entirely unnecessary.

2.5. Closing the Connection

The WebSocket Connection is closed as specified in Section 7 of [RFC6455].

All requests for which the CoAP client has not received a response yet are cancelled when the connection is closed. If the client observes one or more resources over the WebSocket Connection, then the CoAP server (or intermediary in the role of the CoAP server) MUST remove all entries associated with the client from the lists of observers when the connection is closed.

3. CoAP over WebSockets URIs

For the first configuration discussed in Section 1.1, this document defines two new URIs schemes that can be used for identifying CoAP resources and providing a means of locating these resources: "coap+ws" and "coap+wss".

Similar to the "coap" and "coaps" schemes, the "coap+ws" and "coap+wss" schemes organize resources hierarchically under a CoAP origin server. The key difference is that the server is potentially reachable on a WebSocket endpoint instead of a UDP endpoint.

The WebSocket endpoint is identified by a "ws" or "wss" URI that is composed of the authority part of the "coap+ws" or "coap+wss" URI, respectively, and the well-known path `"/.well-known/coap"` [RFC5785]. The path and query parts of a "coap+ws" or "coap+wss" URI identify a resource within the specified endpoint which can be operated on by the methods defined by the CoAP protocol.

The syntax of the "coap+ws" and "coap+wss" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty" and "query" are the same as in [RFC3986].

```
coap-ws-URI =  
    "coap+ws:" "/" host [ ":" port ] path-abempty [ "?" query ]  
  
coap-wss-URI =  
    "coap+wss:" "/" host [ ":" port ] path-abempty [ "?" query ]
```

The port component is OPTIONAL; the default for "coap+ws" is port 80, while the default for "coap+wss" is port 443.

Fragment identifiers are not part of the request URI and thus MUST NOT be transmitted in a WebSocket handshake or in the URI options of a CoAP request.

3.1. Decomposing and Composing URIs

The steps for decomposing a "coap+ws" or "coap+wss" URI into CoAP options are the same as specified in Section 6.4 of [RFC7252] with the following changes:

- o The <scheme> component MUST be "coap+ws" or "coap+wss" when converted to ASCII lowercase.
- o A Uri-Host Option MUST only be included in a request when the <host> component does not equal the uri-host component in the Host header field in the WebSocket handshake.
- o A Uri-Port Option MUST only be included in a request if |port| does not equal the port component in the Host header field in the WebSocket handshake.

The steps to construct a URI from a request's options are changed accordingly.

4. Security Considerations

CoAP over WebSockets and CoAP over TLS-secured WebSockets do not introduce additional security issues beyond CoAP and DTLS-secured CoAP respectively [RFC7252].

The security considerations of [RFC6455] apply.

5. IANA Considerations

[Note to RFC Editor: Please replace XXXX in this section with the RFC number of this specification.]

5.1. URI Scheme Registrations

5.1.1. "coap+ws"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+ws".

URI scheme name.

coap+ws

Status.

Permanent.

URI scheme syntax.

Defined in Section 3 of [RFCXXXX].

URI scheme semantics.

The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol.

Encoding considerations.

The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Interoperability considerations.

None.

Security considerations.

See Section 4 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

5.1.2. "coap+wss"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+wss".

URI scheme name.

coap+wss

Status.

Permanent.

URI scheme syntax.

Defined in Section 3 of [RFCXXXX].

URI scheme semantics.

The "coap+wss" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol secured with Transport Layer Security (TLS).

Encoding considerations.

The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Interoperability considerations.

None.

Security considerations.

See Section 4 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

5.2. WebSocket Subprotocol Registration

This document requests the registration of the subprotocol name "coap" in the WebSocket Subprotocol Name Registry.

Subprotocol Identifier.

coap

Subprotocol Common Name.

Constrained Application Protocol (CoAP)

Subprotocol Definition.

[RFCXXXX]

5.3. Well-Known URI Suffix Registration

This document requests the registration of the Well-Known URI suffix "coap" in the Well-Known URI Registry.

URI suffix.
coap

Change controller.
IETF

Specification document(s).
[RFCXXXX]

Related information.
None.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<http://www.rfc-editor.org/info/rfc5785>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<http://www.rfc-editor.org/info/rfc6455>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

6.2. Informative References

- [I-D.becker-core-coap-sms-gprs]
Becker, M., Li, K., Kuladinithi, K., and T. Poetsch,
"Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-05 (work in progress), August 2014.
- [I-D.dijk-core-sleepy-reqs]
Dijk, E., "Sleepy Devices using CoAP - Requirements",
draft-dijk-core-sleepy-reqs-00 (work in progress), June 2013.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP",
draft-ietf-core-block-20 (work in progress), April 2016.
- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Technologies, Z., and H.
Tschofenig, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-tcp-tls-02 (work in progress), April 2016.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<http://www.rfc-editor.org/info/rfc6454>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.

Appendix A. Examples

This section gives examples for the first two configurations discussed in Section 1.1.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be

as follows. Figure 8 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI `<coap+ws://example.org/sensors/temperature?u=Cel>`, for example, from a resource representation that it retrieved previously.
2. It establishes a WebSocket Connection to the endpoint URI composed of the authority "example.org" and the well-known path `"/.well-known/coap"`, `<ws://example.org/.well-known/coap>`.
3. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.
4. It waits for the server to return a response.
5. The CoAP client uses the connection for further requests, or the connection is closed.

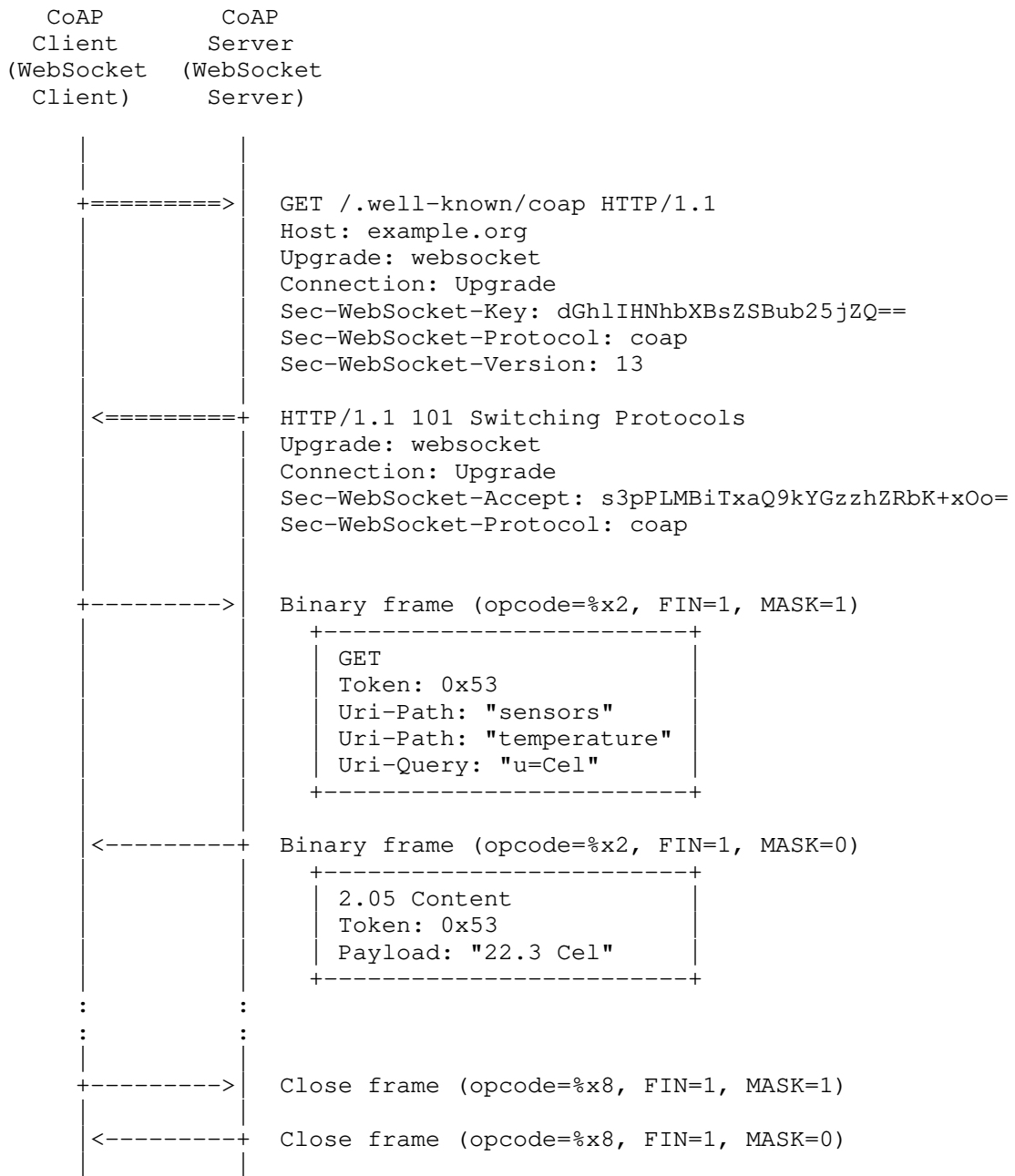


Figure 8: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 9 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource `<coap://[2001:DB8::1]/>`. The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response over the WebSocket connection to the client.

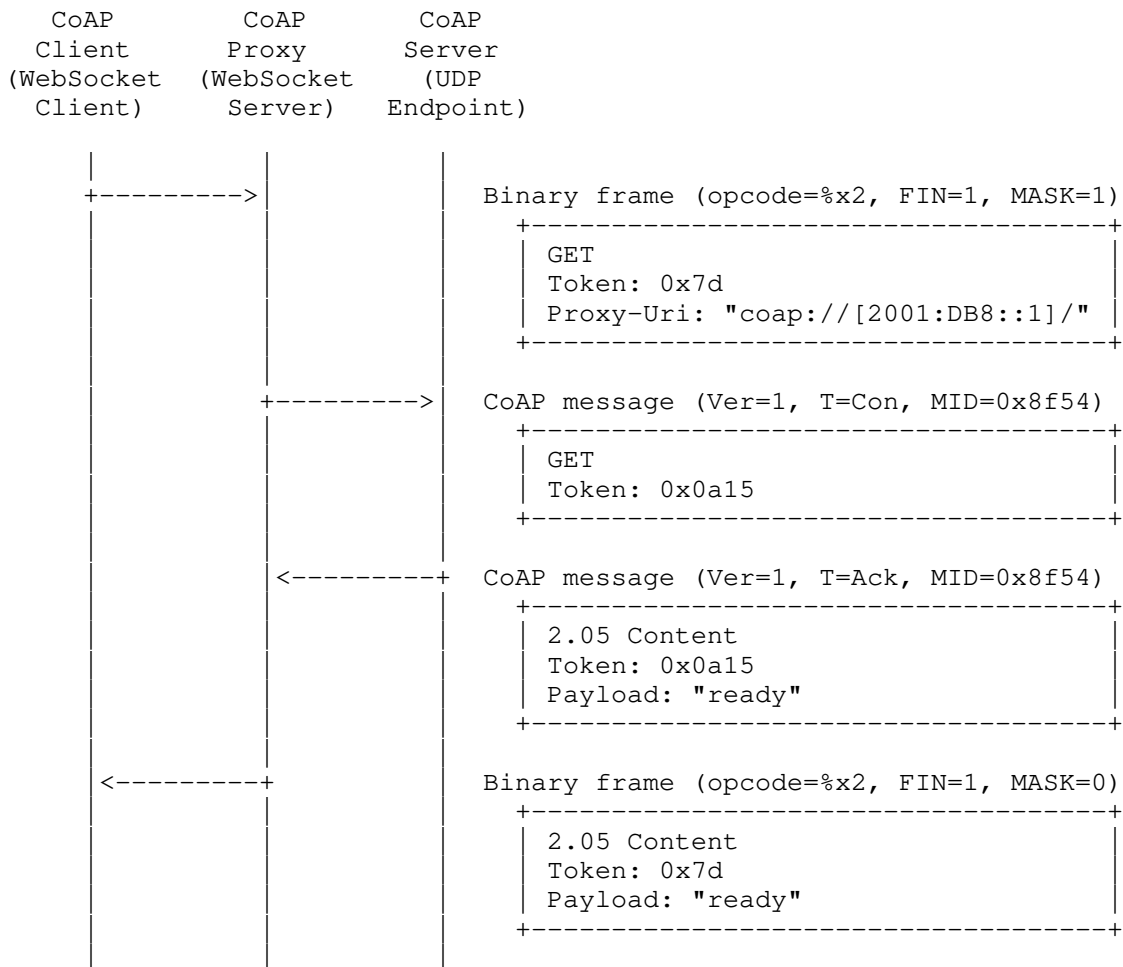


Figure 9: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

Acknowledgements

Thanks to Nadir Javed for helpful comments and discussions that have shaped the document.

Authors' Addresses

Teemu Savolainen
Nokia
Hermiankatu 12 D
Tampere FI-33720
Finland

Email: teemu.savolainen@nokia.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

G. Selander
J. Mattsson
F. Palombini
Ericsson AB
L. Seitz
SICS Swedish ICT
March 21, 2016

Object Security of CoAP (OSCOAP)
draft-selander-ace-object-security-04

Abstract

This memo defines Object Security of CoAP (OSCOAP), a method for application layer protection of message exchanges with the Constrained Application Protocol (CoAP), using the CBOR Encoded Message Syntax. OSCOAP provides end-to-end encryption, integrity and replay protection to CoAP payload, options, and header fields, as well as a secure binding between CoAP request and response messages. The use of OSCOAP is signaled with the CoAP option Object-Security, also defined in this memo.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. The Object-Security Option	5
3. The Security Context	6
4. Protected CoAP Message Fields	8
5. The COSE Object	10
5.1. Plaintext	11
5.2. Additional Authenticated Data	12
6. Protecting CoAP Messages	13
6.1. Replay and Freshness Protection	13
6.2. Protecting the Request	13
6.3. Verifying the Request	14
6.4. Protecting the Response	15
6.5. Verifying the Response	16
7. Security Considerations	16
8. Privacy Considerations	18
9. IANA Considerations	18
9.1. CoAP Option Number Registration	18
9.2. Media Type Registrations	19
9.3. CoAP Content Format Registration	20
10. Acknowledgments	21
11. References	21
11.1. Normative References	21
11.2. Informative References	21
Appendix A. Overhead	22
A.1. Length of the Object-Security Option	22
A.2. Size of the COSE Object	23
A.3. Message Expansion	24
A.4. Example	24
Appendix B. Examples	25
B.1. Secure Access to Actuator	25
B.2. Secure Subscribe to Sensor	27
Appendix C. Object Security of Content (OSCON)	28
C.1. Overhead OSCON	30
C.2. MAC Only	30
C.3. Signature Only	31
C.4. Authenticated Encryption with Additional Data (AEAD)	32
C.5. Symmetric Encryption with Asymmetric Signature (SEAS)	33
Authors' Addresses	33

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web application protocol, designed for constrained nodes and networks [RFC7228]. CoAP specifies the use of proxies, to improve scalability, efficiency, and uses. At the same time CoAP references DTLS [RFC6347] for security. Proxy operations on CoAP messages require DTLS to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or manipulate any part of the CoAP payload and metadata, in transit between client and server. The proxy can also inject, delete, or reorder packages without being protected or detected by DTLS.

This memo defines Object Security of CoAP (OSCOAP), a data object based security protocol, protecting CoAP message exchanges end-to-end, across intermediary nodes. An analysis of end-to-end security for CoAP messages through intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs]; OSCOAP targets the requirements in Sections 3.1 and 3.2.

OSCOAP builds on the CBOR Encoded Message Syntax (COSE) [I-D.ietf-cose-msg], providing end-to-end encryption, integrity, and replay protection. The use of OSCOAP is signaled with the CoAP option Object-Security, also defined in this memo.

OSCOAP transforms an unprotected CoAP message into a protected CoAP message in the following way: the unprotected CoAP message is protected by including payload (if present), certain options, and header fields in a COSE object. The message fields that have been encrypted are removed from the message whereas the Object-Security option and the COSE object are added. We call the result the "protected" CoAP message. Thus OSCOAP is a security protocol based on the exchange of protected CoAP messages (see Figure 1).

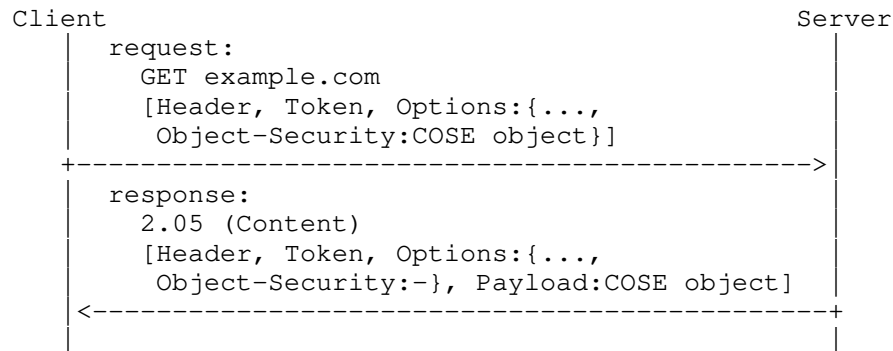


Figure 1: Sketch of OSCOAP

OSCOAP provides protection of CoAP payload, certain options, and header fields, as well as a secure binding between CoAP request and response messages, and freshness of requests and responses.

OSCOAP may be used in constrained settings, where DTLS cannot be supported. Alternatively, OSCOAP can be combined with DTLS, thereby enabling end-to-end security of CoAP payload, in combination with hop-by-hop protection of the entire CoAP message, during transport between end-point and intermediary node. Examples of the use of OSCOAP are given in Appendix B.

The message protection provided by OSCOAP can alternatively be applied to payload only of individual messages. We call this object security of content (OSCON) and it is defined in Appendix C. OSCON targets the requirements in Sections 3.3 – 3.5 of [I-D.hartke-core-e2e-security-reqs].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. These words may also appear in this document in lowercase, absent their normative meanings.

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

Two different scopes of object security are defined:

- o OSCOAP = object security of CoAP, signaled with the Object-Security option.
- o OSCON = object security of content, signaled with Content Format/Media Type set to application/oscon.

OSCON is defined in Appendix C.

2. The Object-Security Option

The Object-Security option indicates that OSCOAP is used to protect the CoAP message exchange.

The Object-Security option is critical, safe to forward, part of the cache key, and not repeatable. Figure 2 illustrates the structure of the Object-Security option.

A CoAP proxy SHOULD NOT cache a response to a request with an Object-Security option, since the response is only applicable to the original client's request. The Object-Security option is included in the cache key for backward compatibility with proxies not recognizing the Object-Security option. The effect of this is that messages with the Object-Security option will never generate cache hits. To further prevent caching, a Max-Age option with value zero can be added to the protected CoAP responses.

No.	C	U	N	R	Name	Format	Length
TBD	x				Object-Security	opaque	0-

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Figure 2: The Object-Security Option

The length of the Object-Security option depends on whether the unprotected message has payload, on the set of options that are included in the unprotected message, the length of the integrity tag, and the length of the information identifying the security context.

- o If the unprotected message has payload, then the COSE object is the payload of the protected message (see Section 6.2 and Section 6.4), and the Object-Security option has length zero.
- o If the unprotected message does not have payload, then the COSE object is the value of the Object-Security option and the length of the Object-Security option is equal to the size of the COSE object.

An example of option length is given in Appendix A.

3. The Security Context

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCOAP. A security context needs to be pre-established and agreed upon between client and server. How this is done is out of scope of this memo, an example is given in the appendices of [I-D.selander-ace-cose-ecdhe]. Each security context is identified by a Context Identifier, which is unique within a given server. A Context Identifier that is no longer in use can be reassigned to a new security context.

The security context has a "Client Write" part and a "Server Write" part. The client initiating a transaction uses the Client Write part of the context to protect the request; the server receiving the request first uses the Client Write part of the context to verify the request, then the Server Write part of the context to protect the response. Finally, the client uses the Server Write part of the context to verify the response.

OSCOAP is very similar to TLS and borrows mechanisms such as key derivation, and nonce construction from [I-D.ietf-tls-tls13]. The main difference is that OSCOAP uses COSE [I-D.ietf-cose-msg] instead of the TLS record layer, which allows OSCOAP to use a context identifier, and sequence numbers of variable length.

It should be noted that how the context is retrieved within the client and server is linked to the resource discovery, may be implementation specific, and is out of scope of this memo.

An example is shown in Figure 3.

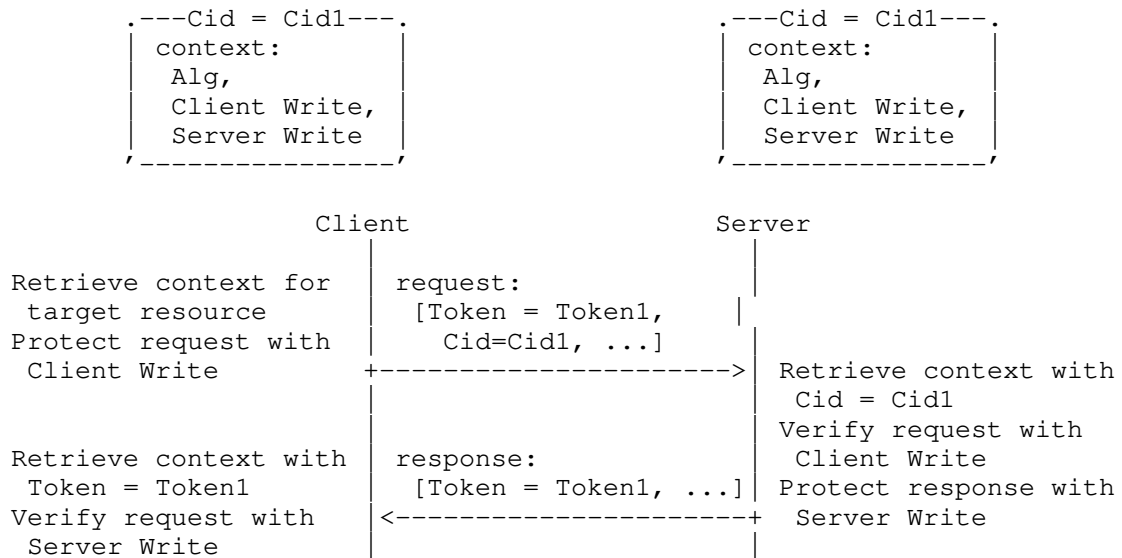


Figure 3: Retrieval and use of the Security Context

The security context structure contains the following parameters:

- o Context Identifier (Cid). Variable length byte string that identifies the security context. Immutable.
- o Algorithm (Alg). Value that identifies the COSE AEAD algorithm to use for encryption. Immutable.
- o Client Write Key. Byte string containing the symmetric key to use in client-sent messages. Length is determined by Algorithm. Immutable.
- o Client Write IV. Byte string containing the static IV to use in cryptographic operations on client-sent messages. Length is determined by Algorithm. Immutable.
- o Client Write Sequence Number. Non-negative integer enumerating the COSE objects that the client sent, associated to the Context Identifier. It is used for replay protection, and to generate unique nonces. Initiated to 0. Maximum value is determined by Algorithm.
- o Server Write Key. Byte string containing the symmetric key to use in server-sent messages. Length is determined by the Algorithm. Immutable.

- o Server Write IV. Byte string containing the static IV to use in cryptographic operations on server-sent messages. Length is determined by Algorithm. Immutable.
- o Server Write Sequence Number. Non-negative integer enumerating the COSE objects that the server sent, associated to the Context Identifier. It is used for replay protection, and to generate unique nonces. Initiated to 0. Maximum value is determined by Algorithm.
- o Replay Window. The replay protection window for messages received, equivalent to the functionality described in Section 4.1.2.6 of [RFC6347]. The default window size is 64.

The size of Cid depends on the number of simultaneous clients, and must be chosen so that the server can uniquely identify the requesting client. Cids of different lengths can be used by different client. In the case of an ACE-based authentication and authorization model [I-D.ietf-ace-oauth-authz], the Authorization Server can define the context identifier of all clients, interacting with a particular server, in which case the size of Cid can be proportional to the logarithm of the number of authorized clients. It is RECOMMENDED to start assigning Cids of length 1 byte (0x00, 0x01, ..., 0xff), and then when all 1 byte Cids are in use, start handling out Cids with a length of two bytes (0x0000, 0x0001, ..., 0xffff), and so on.

The ordered pair (Cid, Client Write Sequence Number) is called Transaction Identifier (Tid), and SHALL be unique for each COSE object and server. The Tid is used as a unique challenge in the COSE object of the protected CoAP request, and in part of the Additional Authenticated Data (AAD, see Section 5) of the protected CoAP response message.

4. Protected CoAP Message Fields

This section defines how the CoAP message fields are protected. OSCOAP protects as much of the unprotected CoAP message as possible, while still allowing forward proxy operations [I-D.hartke-core-e2e-security-reqs].

The CoAP Payload SHALL be encrypted and integrity protected.

The CoAP Header fields Version and Code SHALL be integrity protected but not encrypted. The CoAP Message Layer parameters, Type and Message ID, as well as Token and Token Length SHALL neither be integrity protected nor encrypted.

Protection of CoAP Options can be summarized as follows:

- o To prevent information leakage, Uri-Path and Uri-Query SHALL be encrypted. As a consequence, if Proxy-Uri is used, those parts of the URI SHALL be removed from the Proxy-Uri. The CoAP Options Uri-Host, Uri-Port, Proxy-Uri, and Proxy-Scheme SHALL neither be encrypted, nor integrity protected (cf. protection of request URI in Section 5.2).
- o The other CoAP options listed in Figure 4 SHALL be encrypted and integrity protected.

No.	C	U	N	R	Name	Format	Length	E	I	D
1	x			x	If-Match	opaque	0-8	x	x	
3	x	x	-		Uri-Host	string	1-255			
4				x	ETag	opaque	1-8	x	x	
5	x				If-None-Match	empty	0	x	x	
6		x	-		Observe	uint	0-3	x	x	x
7	x	x	-		Uri-Port	uint	0-2			
8				x	Location-Path	string	0-255	x	x	
11	x	x	-	x	Uri-Path	string	0-255	x	x	
12					Content-Format	uint	0-2	x	x	
14		x	-		Max-Age	uint	0-4	x	x	x
15	x	x	-	x	Uri-Query	string	0-255	x	x	
17	x				Accept	uint	0-2	x	x	
20				x	Location-Query	string	0-255	x	x	
35	x	x	-		Proxy-Uri	string	1-1034			
39	x	x	-		Proxy-Scheme	string	1-255			
60			x		Size1	uint	0-4	x	x	

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt, I=Integrity Protect, D=Duplicate.

Figure 4: Protected CoAP Options

Unless specified otherwise, CoAP options not listed in Figure 4 SHALL be encrypted and integrity protected.

Specifications of new CoAP options SHOULD specify how they are processed with OSCOAP. New COAP options SHOULD be encrypted and integrity protected. New COAP options SHALL be integrity protected unless a proxy needs to change the option, and SHALL be encrypted unless a proxy needs to read the option.

The encrypted options are in general omitted from the protected CoAP message and not visible to intermediary nodes (see Section 6.2 and

Section 6.4). Hence the actions resulting from the use of corresponding options is analogous to the case of communicating directly with the endpoint. For example, a client using an ETag option will not be served by a proxy.

However, some options which are encrypted need to be present in the protected CoAP message to support certain proxy functions. A CoAP option which may be both encrypted in the COSE object of the protected CoAP message, and also unencrypted as CoAP option in the protected CoAP message, is called "duplicate". The "encrypted" value of a duplicate option is intended for the destination endpoint and the "unencrypted" value is intended for a proxy. The unencrypted value is not integrity protected.

- o The Max-Age option is duplicate. The unencrypted Max-Age SHOULD have value zero to prevent caching of responses. The encrypted Max-Age is used as defined in [RFC7252] taking into account that it is not accessible proxies.
- o The Observe option is duplicate. If used, then the encrypted Observe and the unencrypted Observe SHALL have the same value. The Observe option as used here targets the requirements of Section 3.2 of [I-D.hartke-core-e2e-security-reqs].

Specifications of new CoAP options SHOULD specify if the option is duplicate and how it are processed with OSCOAP. New COAP options SHOULD NOT be duplicate.

5. The COSE Object

This section defines how to use the COSE format [I-D.ietf-cose-msg] to wrap and protect data in the unprotected CoAP message. OSCOAP uses the COSE_Encrypted structure with an Authenticated Encryption with Additional Data (AEAD) algorithm.

The mandatory to support AEAD algorithm is AES-CCM-64-64-128 defined in Section 10.2 of [I-D.ietf-cose-msg]. For AES-CCM-64-64-128 the length of Client Write Key and the Server Write Key SHALL be 128 bits, the length of the nonce, Client Write IV, and the Server Write IV SHALL be 7 bytes, and the maximum Client Write Sequence Number and Server Write Sequence Number SHALL be $2^{56}-1$. The nonce is constructed exactly like in Section 5.2.2 of [I-D.ietf-tls-tls13], i.e. by padding the Client Write Sequence Number or the Server Write Sequence Number with zeroes and XORing it with the static Client Write IV or Server Write IV, respectively.

Since OSCOAP only makes use of a single COSE structure, there is no need to explicitly specify the structure, and OSCOAP uses the

untagged version of the COSE_Encrypted structure (Section 2. of [I-D.ietf-cose-msg]). If the COSE object has a different structure, the receiver MUST reject the message, treating it as malformed.

We denote by Plaintext the data that is encrypted and integrity protected, and by Additional Authenticated Data (AAD) the data that is integrity protected only, in the COSE object.

The fields of COSE_Encrypted structure are defined as follows (see example in Appendix C.4).

- o The "Headers" field is formed by:
 - * The "protected" field, which SHALL include:
 - + The "Partial Initialization Vector" parameter. The value is set to the Client Write Sequence Number, or the Server Write Sequence Number, depending on whether the client or server is sending the message. The Partial IV is a byte string (type: bstr), where the length is the minimum length needed to encode the sequence number.
 - + If the message is a CoAP request, the "kid" parameter. The value is set to the Context Identifier (see Section 3).
 - * The "unprotected" field, which SHALL be empty.
- o The "ciphertext" field is computed from the Plaintext and the Additional Authenticated Data (AAD) and encoded as a byte string (type: bstr), following Section 5.2 of [I-D.ietf-cose-msg].

5.1. Plaintext

The Plaintext is formatted as a CoAP message without Header (see Figure 5) consisting of:

- o all CoAP Options present in the unprotected message which are encrypted (see Section 4), in the order as given by the Option number (each Option with Option Header including delta to previous included encrypted option); and
- o the CoAP Payload, if present, and in that case prefixed by the one-byte Payload Marker (0xFF).

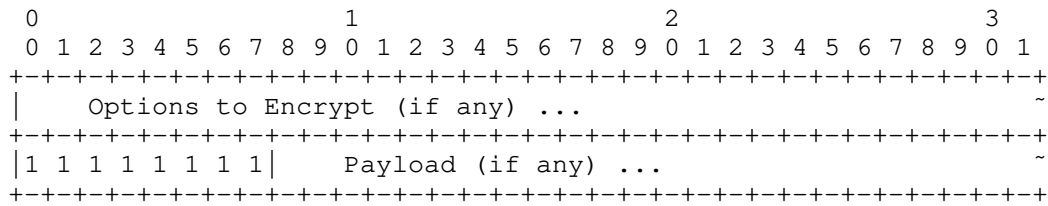


Figure 5: Plaintext

5.2. Additional Authenticated Data

The Additional Authenticated Data ("Enc_structure") as described in Section 5.3 of [I-D.ietf-cose-msg] includes (see Figure 6):

- o the "context" parameter, which has value "Encrypted"
- o the "protected" parameter, which includes the "protected" part of the "Headers" field;
- o the "external_aad" includes:
 - * the two first bytes of the CoAP header in the unprotected message (including Version and Code) with Type and Token Length bits set to 0;
 - * The Algorithm from the security context used for the exchange;
 - * the plaintext request URI composed from the request scheme and Uri-* options according to the method described in Section 6.5 of [RFC7252], if the message is a CoAP request; and
 - * the Transaction Identifier (Tid) of the associated CoAP request, if the message is a CoAP response (see Section 3).

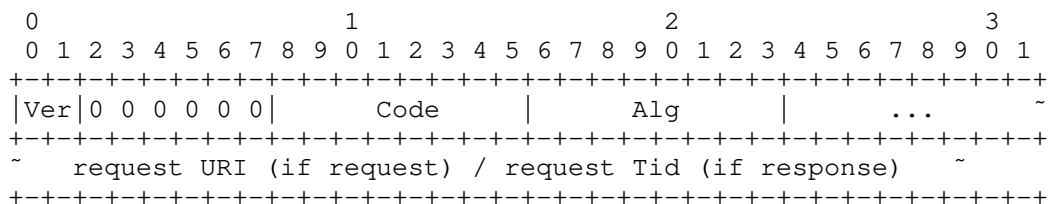


Figure 6: Additional Authenticated Data

The encryption process is described in Section 5.3 of [I-D.ietf-cose-msg].

6. Protecting CoAP Messages

6.1. Replay and Freshness Protection

In order to protect from replay of messages and verify freshness, a CoAP endpoint SHALL maintain a Client Write Sequence Number, and a Server Write Sequence Number associated to a security context, which is identified with a Context Identifier (Cid). The two sequence numbers are the highest sequence number the endpoint has sent and the highest sequence number the endpoint has received. A client uses the Client Write Sequence Number for protecting sent messages and the Server Write Sequence Number for verifying received messages, and vice versa for the server, as described in Section 3.

Depending on use case and ordering of messages provided by underlying layers, an endpoint MAY maintain a sliding replay window for Sequence Numbers of received messages associated to each Cid.

A receiving endpoint SHALL verify that the Sequence Number received in the COSE object has not been received before in the security context identified by the Cid. Note that for the server, the relevant Sequence Number here is the Client Write Sequence Number and vice versa for the client.

OSCOAP is a challenge-response protocol, where the response is verified to match a prior request, by including the unique transaction identifier (Tid as defined in Section 3) of the request in the Additional Authenticated Data of the response message.

If a CoAP server receives a request with the Object-Security option, then the server SHALL include the Tid of the request in the AAD of the response, as described in Section 6.4.

If the CoAP client receives a response with the Object-Security option, then the client SHALL verify the integrity of the response, using the Tid of its own associated request in the AAD, as described in Section 6.5.

6.2. Protecting the Request

Given an unprotected CoAP request, including header, options and payload, the client SHALL perform the following steps to create a protected CoAP request using a security context associated with the target resource:

1. Increment the Client Write Sequence Number by one (note that this means that sequence number 0 is never used). If the Client Write Sequence Number exceeds the maximum number for the AEAD

algorithm, the client MUST NOT process any requests with the given security context. The client SHOULD acquire a new security context before this happens. The latter is out of scope of this memo.

2. Compute the COSE object as specified in Section 5
 - * the nonce in the AEAD is created by XORing the static IV (Client Write IV) with the partial IV (Client Write Sequence Number).
3. Format the protected CoAP message as an ordinary CoAP message, with the following Header, Options, and Payload, based on the unprotected CoAP message:
 - * The CoAP header is the same as the unprotected CoAP message.
 - * The CoAP options which are encrypted and not duplicate (Section 4) are removed. Any duplicate option which is present has its unencrypted value. The Object-Security option is added.
 - * If the unprotected CoAP message has no Payload, then the value of the Object-Security option is the COSE object. If the unprotected CoAP message has Payload, then the Object-Security option is empty and the Payload of the protected CoAP message is the COSE object.

The Client SHALL be able to find the correct security context with use of the Token of the message exchange.

6.3. Verifying the Request

A CoAP server receiving a message containing the Object-Security option SHALL perform the following steps, using the security context identified by the Context Identifier in the "kid" parameter in the received COSE object:

1. Verify the Sequence Number in the Partial IV parameter, as described in Section 6.1. If it cannot be verified that the Sequence Number has not been received before, the server MUST stop processing the request.
2. Recreate the Additional Authenticated Data, as described in Section 5.
3. Compose the nonce by XORing the static IV (Client Write IV) with the Partial IV parameter, received in the COSE Object.

4. Retrieve the Client Write Key.
5. Verify and decrypt the message. If the verification fails, the server MUST stop processing the request.
6. If the message verifies, update the Client Write Sequence Number or Replay Window, as described in Section 6.1.
7. Restore the unprotected request by adding any decrypted options or payload from the plaintext. Any duplicate options (Section 4) are overwritten. The Object-Security option is removed.

6.4. Protecting the Response

A server receiving a valid request with a protected CoAP message (i.e. containing an Object-Security option) SHALL respond with a protected CoAP message.

Given an unprotected CoAP response, including header, options, and payload, the server SHALL perform the following steps to create a protected CoAP response, using the security context identified by the Context Identifier of the received request:

1. Increment the Server Write Sequence Number by one (note that this means that sequence number 0 is never used). If the Server Write Sequence Number exceeds the maximum number for the AEAD algorithm, the server MUST NOT process any more responses with the given security context. The server SHOULD acquire a new security context before this happens. The latter is out of scope of this memo.
2. Compute the COSE object as specified in Section 5
 - * The nonce in the AEAD is created by XORing the static IV (Server Write IV) and the Server Write Sequence Number.
3. Format the protected CoAP message as an ordinary CoAP message, with the following Header, Options, and Payload based on the unprotected CoAP message:
 - * The CoAP header is the same as the unprotected CoAP message.
 - * The CoAP options which are encrypted and not duplicate (Section 4) are removed. Any duplicate option which is present has its unencrypted value. The Object-Security option is added.

- * If the unprotected CoAP message has no Payload, then the value of the Object-Security option is the COSE object. If the unprotected CoAP message has Payload, then the Object-Security option is empty, and the Payload of the protected CoAP message is the COSE object.

Note the differences between generating a protected request, and a protected response, for example whether "kid" is present in the header, or whether Destination URI or Tid is present in the AAD, of the COSE object.

6.5. Verifying the Response

A CoAP client receiving a message containing the Object-Security option SHALL perform the following steps, using the security context identified by the Token of the received response:

1. Verify the Sequence Number in the Partial IV parameter as described in Section 6.1. If it cannot be verified that the Sequence Number has not been received before, the client MUST stop processing the response.
2. Recreate the Additional Authenticated Data as described in Section 5.
3. Compose the nonce by XORing the static IV (Server Write IV) with the Partial IV parameter, received in the COSE Object.
4. Retrieve the Server Write Key.
5. Verify and decrypt the message. If the verification fails, the client MUST stop processing the response.
6. If the message verifies, update the Client Write Sequence Number or Replay Window, as described in Section 6.1.
7. Restore the unprotected response by adding any decrypted options or payload from the plaintext. Any duplicate options (Section 4) are overwritten. The Object-Security option is removed.

7. Security Considerations

In scenarios with intermediary nodes such as proxies or brokers, transport layer security such as DTLS only protects data hop-by-hop. As a consequence the intermediary nodes can read and modify information. The trust model where all intermediate nodes are considered trustworthy is problematic, not only from a privacy perspective, but also from a security perspective, as the

intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases, where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture brittle.

DTLS protects hop-by-hop the entire CoAP message, including header, options, and payload. OSCOAP protects end-to-end the payload, and all information in the options and header, that is not required for forwarding (see Section 4). DTLS and OSCOAP can be combined.

The CoAP message layer, however, cannot be protected end-to-end through intermediary devices since the parameters Type and Message ID, as well as Token and Token Length may be changed by a proxy. Moreover, messages that are not possible to verify should for security reasons not always be acknowledged but in some cases be silently dropped. This would not comply with CoAP message layer, but does not have an impact on the application layer security solution, since message layer is excluded from that.

The specification in this memo assumes that there is an established security context. [I-D.ietf-ace-oauth-authz] presents a method for a trusted third party (Authorization Server) to enable key establishment between potentially constrained nodes, using OAuth and PoP Tokens. [I-D.selander-ace-cose-ecdh] describes a Diffie-Hellman key exchange, authenticated with pre-established keys, and a key derivation method for producing a security context, suitable for OSCOAP. The two methods can be combined, enabling a client and server with relation to a trusted third party to establish a security context with forward secrecy.

For symmetric encryption it is required to have a unique nonce for each message, for which the sequence numbers in the COSE message field "Partial IV" is used. The nonce SHALL be the XOR of a static IV and the sequence number. The static IVs (Client Write IV and Server Write IV) SHOULD be established between sender and receiver before the message is sent, to avoid the overhead of sending it in each message, for example using the method in [I-D.selander-ace-cose-ecdh].

As the receiver accepts any sequence number larger than the one previously received, the problem of sequence number synchronization is avoided. The alternatives have issues: very constrained devices may not be able to support accurate time, or to generate and store large numbers of random nonces. The requirement to change key at counter wrap is a complication, but it also forces the user of this specification to think about implementing key renewal.

Block-wise transfers as currently defined in [I-D.ietf-core-block] cannot be protected end-to-end because the payload as well as the Block1/Block2 options may be changed in an unpredictable way by a proxy. Since [I-D.ietf-core-block] allows for any proxy to fragment the payload, an endpoint receiving a message fragment with a block option is not able to verify integrity of that fragment. As a consequence, block-wise disables end-to-end security: an adversary may inject an unlimited number of messages with a block option claiming it to be a sequence of message fragments without the receiving endpoint being able to disprove the claim.

If instead the payload and block options Block1/Block2 were not allowed to be changed by intermediate devices, then the message fragments could be integrity protected end-to-end. In that case each individual block can be securely verified by the receiver, retransmission securely requested etc. Since the blocks are enumerated sequentially, and carry information about whether this fragment is the last, when all blocks have been securely received is enough to prove that the entire message has been securely transferred.

8. Privacy Considerations

Privacy threats executed through intermediate nodes are considerably reduced by means of OSCOAP. End-to-end integrity protection and encryption of CoAP payload and all options that are not used for forwarding, provides mitigation against attacks on sensor and actuator communication, which may have a direct impact the personal sphere.

CoAP headers sent in plaintext allow for example matching of CON and ACK (CoAP Message Identifier), matching of request and responses (Token) and traffic analysis.

9. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[[this document]]" with the RFC number of this specification.

9.1. CoAP Option Number Registration

The Object-Security option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	Object-Security	[[this document]]

9.2. Media Type Registrations

The "application/oscon" media type is added to the Media Types registry:

Type name: application

Subtype name: cose

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section of [[this document]].

Interoperability considerations: N/A

Published specification: [[this document]]

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

- * Magic number(s): N/A

- * File extension(s): N/A

- * Macintosh file type code(s): N/A

Person & email address to contact for further information:
iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander, goran.selander@ericsson.com

Change Controller: IESG

Provisional registration? No

9.3. CoAP Content Format Registration

The "application/oscon" content format is added to the CoAP Content Format registry:

Media type	Encoding	ID	Reference
application/oscon	-	70	[[this document]]

10. Acknowledgments

Klaus Hartke has independently been working on the same problem and a similar solution: establishing end-to-end security across proxies by adding a CoAP option. We are grateful to Malisa Vucinic for providing helpful and timely reviews of previous versions of the draft.

11. References

11.1. Normative References

- [I-D.ietf-cose-msg]
 Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-10 (work in progress), February 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

11.2. Informative References

- [I-D.hartke-core-e2e-security-reqs]
 Selander, G., Palombini, F., Hartke, K., and L. Seitz, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-00 (work in progress), March 2016.

[I-D.ietf-ace-oauth-authz]

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authorization for the Internet of Things using OAuth 2.0", draft-ietf-ace-oauth-authz-01 (work in progress), February 2016.

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

[I-D.ietf-tls-tls13]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-11 (work in progress), December 2015.

[I-D.selander-ace-cose-ecdh]

Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdh-00 (work in progress), March 2016.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.

Appendix A. Overhead

OSCOAP transforms an unprotected CoAP message to a protected CoAP message, and the protected CoAP message is larger than the unprotected CoAP message. This appendix illustrates the message expansion.

A.1. Length of the Object-Security Option

The protected CoAP message contains the COSE object. The COSE object is included in the payload if the unprotected CoAP message has payload or else in the Object-Security option. In the former case the Object-Security option is empty. So the length of the Object-Security option is either zero or the size of the COSE object, depending on whether the CoAP message has payload or not.

Length of Object-Security option = { 0, size of COSE Object }

A.2. Size of the COSE Object

The size of the COSE object is the sum of the sizes of

- o the Header parameters,
- o the Ciphertext (excluding the Tag),
- o the Tag, and
- o data incurred by the COSE format itself (including CBOR encoding).

Let's analyse the contributions one at a time:

- o The header parameters of the COSE object are the Context Identifier (Cid) and the Sequence Number (Seq) (also known as the Transaction Identifier (Tid)) if the message is a request, and Seq only if the message is a response (see Section 5).
 - * The size of Cid depends on the number of simultaneous clients, and must be chosen so that the server can uniquely identify the requesting client. For example, in the case of an ACE-based authentication and authorization model [I-D.ietf-ace-oauth-authz], the Authorization Server or the server itself can define the context identifier of all clients interacting with a particular server, in which case the size of Cid can be proportional to the logarithm of number of authorized clients.
 - + As Cids of different lengths can be used by different client, it is RECOMMENDED to start assigning Cids of length 1 byte (0x00, 0x01, ..., 0xff), and then when all 1 byte Cids are in use, start handling out Cids with a length of two bytes (0x0000, 0x0001, ..., 0xffff).
 - * The size of Seq is variable, and increases with the number of messages exchanged.
 - * As the nonce is generated from the padded Sequence Number and a previously agreed upon static IV it is not required to send the whole nonce in the message.
- o The Ciphertext, excluding the Tag, is the encryption of the payload and the encrypted options Section 4, which are present in the unprotected CoAP message.
- o The size of the Tag depends on the Algorithm. For the OSCOAP mandatory algorithm AES-CCM-64-64-128, the Tag is 8 bytes.

- o The overhead from the COSE format itself depends on the sizes of the previous fields, and is of the order of 10 bytes.

A.3. Message Expansion

The message expansion is not the size of the COSE object. The ciphertext in the COSE object is encrypted payload and options of the unprotected CoAP message – the plaintext of which is removed from the protected CoAP message. Since the size of the ciphertext is the same as the corresponding plaintext, there is no message expansion due to encryption; payload and options are just represented in a different way in the protected CoAP message:

- o The encrypted payload is in the payload of the protected CoAP message
- o The encrypted options are in the Object-Security option or within the payload.

Therefore the OSCOAP message expansion is due to Cid (if present), Seq, Tag, and COSE overhead:

$$\text{Message Overhead} = \text{Cid} + \text{Seq} + \text{Tag} + \text{COSE Overhead}$$

Figure 7: OSCOAP message expansion

A.4. Example

This section gives an example of message expansion in a request with OSCOAP.

In this example we assume an extreme 4-byte Cid, based on the assumption of an ACE deployment with billions of clients requesting access to this particular server. (A typical Cid, will be 1-2 byte as is discussed in Appendix A.2.)

- o Cid: 0xa1534e3c

In the example the sequence number is 225, requiring 1 byte to encode. (The size of Seq could be larger depending on how many messages that has been sent as is discussed in Appendix A.2.)

- o Seq: 225

The example is based on AES-CCM-64-64-128.

- o Tag is 8 bytes

The COSE object is represented in Figure 8 using CBOR's diagnostic notation.

```
[
  h'a20444a1534e3c0641e2', # protected:
                             {04:h'a1534e3c',
                              06:h'e2'}
  {},                       # unprotected: -
  Tag                       # ciphertext + 8 byte authentication tag
]
```

Figure 8: Example of message expansion

Note that the encrypted CoAP options and payload are omitted since we target the message expansion (see Appendix A.3). Therefore the size of the COSE Ciphertext equals the size of the Tag, which is 8 bytes.

The COSE object encodes to a total size of 22 bytes, which is the message expansion in this example. The COSE overhead in this example is $22 - (4 + 1 + 8) = 9$ bytes, according to the formula in Figure 7. Note that in this example two bytes in the COSE overhead are used to encode the length of Cid and the length of Seq.

Figure 9 summarizes these results.

Tid	Tag	COSE OH	Message OH
5 bytes	8 bytes	9 bytes	22 bytes

Figure 9: Message overhead for a 5-byte Tid and 8-byte Tag.

Appendix B. Examples

This section gives examples of OSCOAP. The message exchanges are made, based on the assumption that there is a security context established between client and server. For simplicity, these examples only indicate the content of the messages without going into detail of the COSE message format.

B.1. Secure Access to Actuator

Here is an example targeting the scenario in Section 3.1 of [I-D.hartke-core-e2e-security-reqs]. The example illustrates a client requesting valve 34 to be turned to position 3 (PUT /valve34 with payload value "3"), and getting a confirmation. The CoAP options Uri-Path and Payload are encrypted and integrity protected,

and the CoAP header field Code is integrity protected (see Section 4).

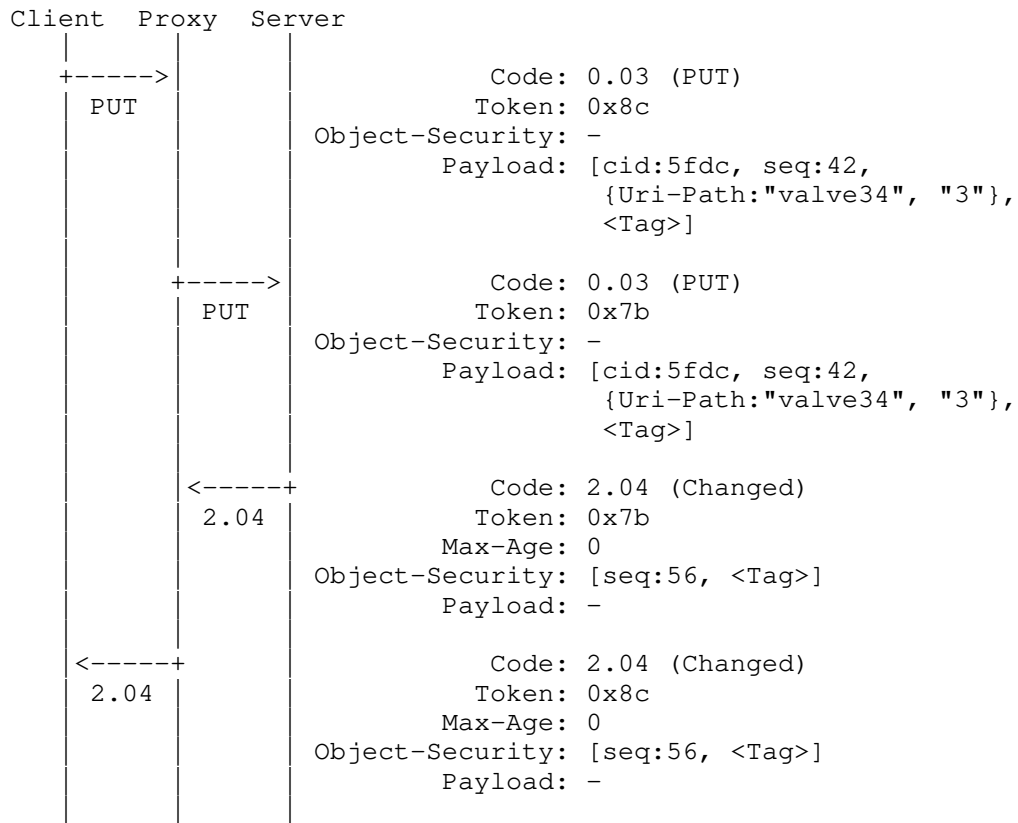


Figure 10: Indication of CoAP PUT protected with OSCOAP. The brackets [...] indicate a COSE object. The brackets { ... } indicate encrypted data.

Since the unprotected request message (PUT) has payload ("3"), the COSE object (indicated with [...]) is carried as the CoAP payload. Since the unprotected response message (Changed) has no payload, the Object-Security option carries the COSE object as its value.

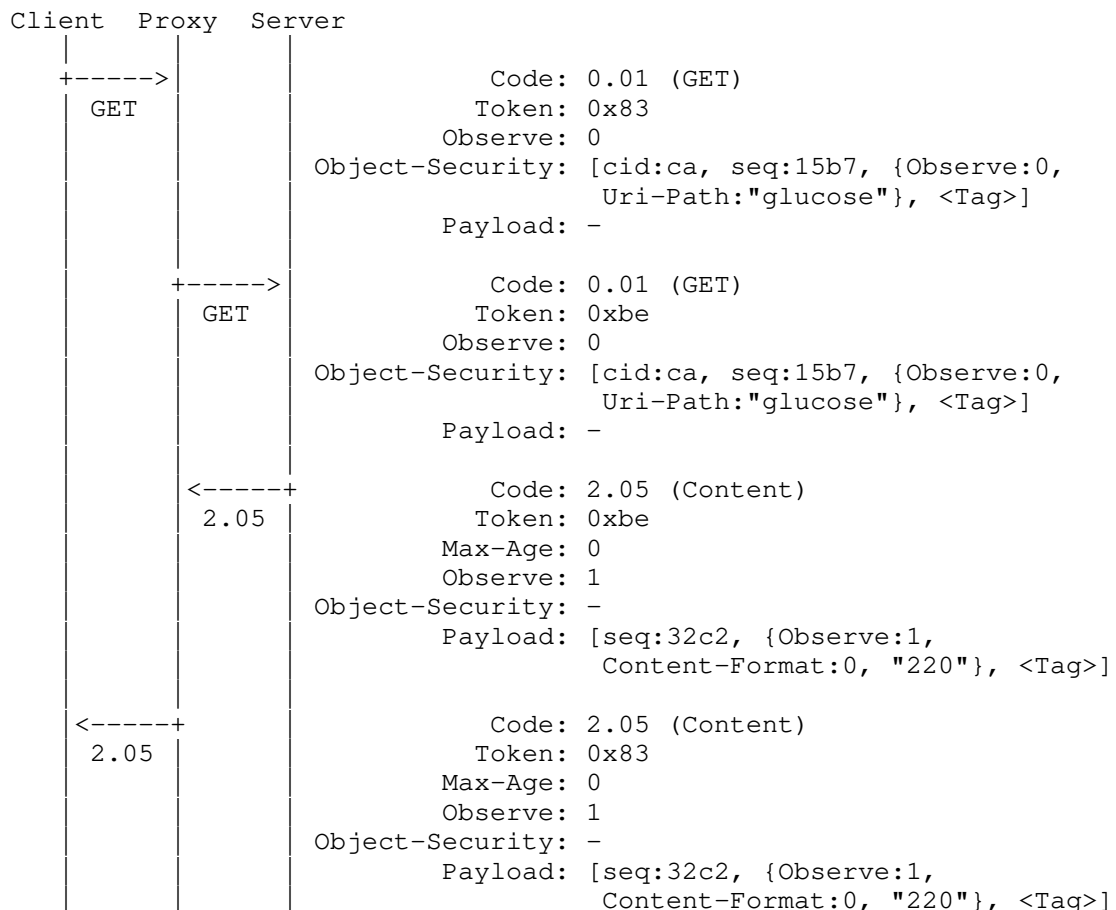
The COSE header of the request contains a Context Identifier (cid:5fdc), indicating which security context was used to protect the message and a Sequence Number (seq:42).

The option Uri-Path (valve34) and payload ("3") are formatted as indicated in Section 5, and encrypted in the COSE Ciphertext (indicated with { ... }).

The server verifies that the Sequence Number has not been received before (see Section 6.1). The client verifies that the Sequence Number has not been received before and that the response message is generated as a response to the sent request message (see Section 6.1).

B.2. Secure Subscribe to Sensor

Here is an example targeting the scenario in Section 3.2 of [I-D.hartke-core-e2e-security-reqs]. The example illustrates a client requesting subscription to a blood sugar measurement resource (GET /glucose), and first receiving the value 220 mg/dl, and then a second reading with value 180 mg/dl. The CoAP options Observe, Uri-Path, Content-Format, and Payload are encrypted and integrity protected, and the CoAP header field Code is integrity protected (see Section 4).



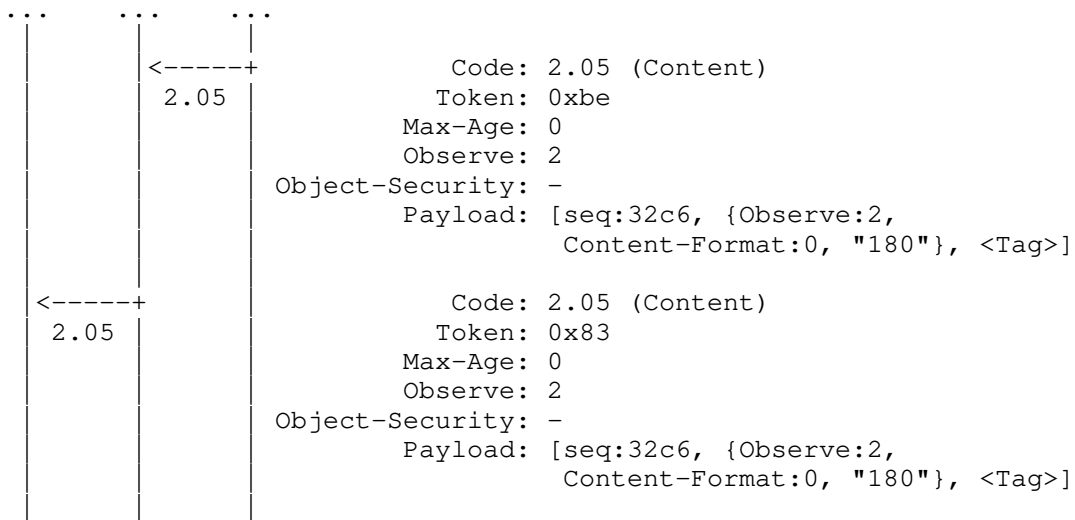


Figure 11: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicates COSE object. The bracket { ... } indicates encrypted data.

Since the unprotected request message (GET) has no payload, the COSE object (indicated with [...]) is carried in the Object-Security option value. Since the unprotected response message (Content) has payload, the Object-Security option is empty, and the COSE object is carried as the payload.

The COSE header of the request contains a Context Identifier (cid:ca), indicating which security context was used to protect the message and a Sequence Number (seq:15b7).

The options Observe, Content-Format and the payload are formatted as indicated in Section 5, and encrypted in the COSE ciphertext (indicated with { ... }).

The server verifies that the Sequence Number has not been received before (see Section 6.1). The client verifies that the Sequence Number has not been received before and that the response message is generated as a response to the subscribe request.

Appendix C. Object Security of Content (OSCON)

OSCOAP protects message exchanges end-to-end between a certain client and a certain server, targeting the security requirements in Section 3.1 and 3.2 of [I-D.hartke-core-e2e-security-reqs]. In contrast, many use cases require one and the same message to be

protected for, and verified by, multiple endpoints, see Sections 3.3 - 3.5 of [I-D.hartke-core-e2e-security-reqs]. Those security requirements can be addressed by protecting essentially the payload/content of individual messages using the COSE format ([I-D.ietf-cose-msg]), rather than the entire request/response message exchange. This is referred to as Object Security of Content (OSCON).

OSCON transforms an unprotected CoAP message into a protected CoAP message in the following way: the payload of the unprotected CoAP message is wrapped by a COSE object, which replaces the payload of the unprotected CoAP message. We call the result the "protected" CoAP message.

The unprotected payload SHALL be the plaintext/payload of the COSE object. The 'protected' field of the COSE object 'Headers' SHALL include the context identifier, both for requests and responses. If the unprotected CoAP message includes a Content-Format option, then the COSE object SHALL include a protected 'content type' field, whose value is set to the unprotected message Content-Format value. The Content-Format option of the protected CoAP message SHALL be replaced with "application/oscon" (Section 9)

The COSE object SHALL be protected (encrypted) and verified (decrypted) as described in ([I-D.ietf-cose-msg]).

In the case of symmetric encryption, the same key and nonce SHALL NOT be used twice. The use of sequence numbers for partial IV as specified for OSCOAP MAY be used. of sequence numbers for replay protection as described in Section 6.1 MAY be used. The use of time stamps in the COSE header parameter 'operation time' [I-D.ietf-cose-msg] for freshness MAY be used.

OSCON SHALL NOT be used in cases where CoAP header fields (such as Code or Version) or CoAP options need to be integrity protected or encrypted. OSCON SHALL NOT be used in cases which require a secure binding between request and response.

The scenarios in Sections 3.3 - 3.5 of [I-D.hartke-core-e2e-security-reqs] assume multiple receivers for a particular content. In this case the use of symmetric keys does not provide data origin authentication. Therefore the COSE object SHOULD in general be protected with a digital signature.

C.1. Overhead OSCON

In general there are four different kinds of ciphersuites that need to be supported: message authentication code, digital signature, authenticated encryption, and symmetric encryption + digital signature. The use of digital signature is necessary for applications with many legitimate recipients of a given message, and where data origin authentication is required.

To distinguish between these different cases, the tagged structures of COSE are used (see Section 2 of [I-D.ietf-cose-msg]).

The size of the COSE message for selected algorithms are detailed in this section.

The size of the header is shown separately from the size of the MAC/signature. A 4-byte Context Identifier and a 1-byte Sequence Number are used throughout all examples, with these values:

- o Cid: 0xa1534e3c
- o Seq: 0xa3

For each scheme, we indicate the fixed length of these two parameters ("Cid+Seq" column) and of the Tag ("MAC"/"SIG"/"TAG"). The "Message OH" column shows the total expansions of the CoAP message size, while the "COSE OH" column is calculated from the previous columns following the formula in Figure 7.

Overhead incurring from CBOR encoding is also included in the COSE overhead count.

To make it easier to read, COSE objects are represented using CBOR's diagnostic notation rather than a binary dump.

C.2. MAC Only

This example is based on HMAC-SHA256, with truncation to 8 bytes (HMAC 256/64).

Since the key is implicitly known by the recipient, the COSE_Mac0_Tagged structure is used (Section 6.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```

996(                                     # COSE_Mac0_Tagged
[
    h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                06:h'a3'}
    {},                         # unprotected
    h'',                       # payload
    MAC                        # truncated 8-byte MAC
]
)

```

This COSE object encodes to a total size of 26 bytes.

Figure 12 summarizes these results.

Structure	Tid	MAC	COSE OH	Message OH
COSE_Mac0_Tagged	5 B	8 B	13 B	26 B

Figure 12: Message overhead for a 5-byte Tid using HMAC 256/64

C.3. Signature Only

This example is based on ECDSA, with a signature of 64 bytes.

Since only one signature is used, the COSE_Sign1_Tagged structure is used (Section 4.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```

997(                                     # COSE_Sign1_Tagged
[
    h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                06:h'a3'}
    {},                         # unprotected
    h'',                       # payload
    SIG                        # 64-byte signature
]
)

```

This COSE object encodes to a total size of 83 bytes.

Figure 13 summarizes these results.

Structure	Tid	SIG	COSE OH	Message OH
COSE_Sign1_Tagged	5 B	64 B	14 B	83 bytes

Figure 13: Message overhead for a 5-byte Tid using 64 byte ECDSA signature.

C.4. Authenticated Encryption with Additional Data (AEAD)

This example is based on AES-CCM with the MAC truncated to 8 bytes.

It is assumed that the nonce is generated from the Sequence Number and some previously agreed upon static IV. This means it is not required to explicitly send the whole nonce in the message.

Since the key is implicitly known by the recipient, the COSE_Encrypted_Tagged structure is used (Section 5.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```

993(                                     # COSE_Encrypted_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                06:h'a3'}
    {},                         # unprotected
    TAG                         # ciphertext + truncated 8-byte TAG
  ]
)

```

This COSE object encodes to a total size of 25 bytes.

Figure 14 summarizes these results.

Structure	Tid	TAG	COSE OH	Message OH
COSE_Encrypted_Tagged	5 B	8 B	12 B	25 bytes

Figure 14: Message overhead for a 5-byte Tid using AES_128_CCM_8.

C.5. Symmetric Encryption with Asymmetric Signature (SEAS)

This example is based on AES-CCM and ECDSA with 64 bytes signature. The same assumption on the security context as in Appendix C.4. COSE defines the field 'counter signature' that is used here to sign a COSE_Encrypted_Tagged message (see Section 3 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```

993(                                     # COSE_Encrypted_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                06:h'a3'}
    {7:SIG},                     # unprotected:
                                07: 64 bytes signature
    TAG                          # ciphertext + truncated 8-byte TAG
  ]
)
```

This COSE object encodes to a total size of 92 bytes.

Figure 15 summarizes these results.

Structure	Tid	TAG	SIG	COSE OH	Message OH
COSE_Encrypted_Tagged	5 B	8 B	64 B	15 B	92 B

Figure 15: Message overhead for a 5-byte Tid using AES-CCM countersigned with ECDSA.

Authors' Addresses

Goeran Selander
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Ludwig Seitz
SICS Swedish ICT
Scheelevagen 17
Lund 22370
Sweden

Email: ludwig@sics.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 14, 2017

G. Selander
J. Mattsson
F. Palombini
Ericsson AB
L. Seitz
SICS Swedish ICT
October 11, 2016

Object Security of CoAP (OSCOAP)
draft-selander-ace-object-security-06

Abstract

This memo defines Object Security of CoAP (OSCOAP), a method for application layer protection of message exchanges with the Constrained Application Protocol (CoAP), using the CBOR Object Signing and Encryption (COSE) format. OSCOAP provides end-to-end encryption, integrity and replay protection to CoAP payload, options, and header fields, as well as a secure binding between CoAP request and response messages. The use of OSCOAP is signaled with the CoAP option Object-Security, also defined in this memo.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 14, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. The Object-Security Option	5
3. The Security Context	6
3.1. Security Context Definition	6
3.2. Security Context Establishment	9
3.2.1. Derivation of Sender Key/IV, Recipient Key/IV	9
3.2.2. Sequence Numbers and Replay Window	10
3.2.3. Context Identifier and Sender/Recipient ID	10
4. Protected CoAP Message Fields	11
5. The COSE Object	13
5.1. Plaintext	15
5.2. Additional Authenticated Data	15
6. Protecting CoAP Messages	17
6.1. Replay and Freshness Protection	17
6.2. Protecting the Request	18
6.3. Verifying the Request	19
6.4. Protecting the Response	20
6.5. Verifying the Response	21
7. Security Considerations	21
8. Privacy Considerations	23
9. IANA Considerations	23
9.1. Sid Registration	24
9.2. CoAP Option Number Registration	24
9.3. Media Type Registrations	24
9.4. CoAP Content Format Registration	25
10. Acknowledgments	26
11. References	26
11.1. Normative References	26
11.2. Informative References	27
Appendix A. Overhead	28
A.1. Length of the Object-Security Option	28
A.2. Size of the COSE Object	28
A.3. Message Expansion	29
A.4. Example	29
Appendix B. Examples	30
B.1. Secure Access to Sensor	31
B.2. Secure Subscribe to Sensor	32
Appendix C. Object Security of Content (OSCON)	34

C.1. Overhead OSCON	35
C.2. MAC Only	35
C.3. Signature Only	36
C.4. Authenticated Encryption with Additional Data (AEAD)	37
C.5. Symmetric Encryption with Asymmetric Signature (SEAS)	38
Authors' Addresses	38

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web application protocol, designed for constrained nodes and networks [RFC7228]. CoAP specifies the use of proxies for scalability and efficiency. At the same time CoAP references DTLS [RFC6347] for security. Proxy operations on CoAP messages require DTLS to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or manipulate any part of the CoAP payload and metadata, in transit between client and server. The proxy can also inject, delete, or reorder packages without being protected or detected by DTLS.

This memo defines Object Security of CoAP (OSCOAP), a data object based security protocol, protecting CoAP message exchanges end-to-end, across intermediary nodes. An analysis of end-to-end security for CoAP messages through intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs], this specification addresses the forwarding case.

The solution provides an in-layer security protocol for CoAP which does not depend on underlying layers and is therefore favorable for providing security for "CoAP over foo", e.g. CoAP messages passing over both unreliable and reliable transport [I-D.ietf-core-coap-tcp-tls], CoAP over IEEE 802.15.4 IE [I-D.bormann-6lo-coap-802-15-ie].

OSCOAP builds on CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg], providing end-to-end encryption, integrity, and replay protection. The use of OSCOAP is signaled with the CoAP option Object-Security, also defined in this memo. The solution transforms an unprotected CoAP message into a protected CoAP message in the following way: the unprotected CoAP message is protected by including payload (if present), certain options, and header fields in a COSE object. The message fields that have been encrypted are removed from the message whereas the Object-Security option and the COSE object are added. We call the result the "protected" CoAP message. Thus OSCOAP is a security protocol based on the exchange of protected CoAP messages (see Figure 1).

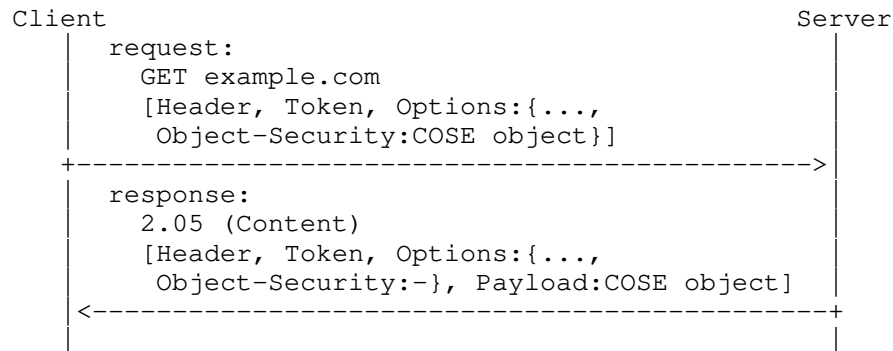


Figure 1: Sketch of OSCOAP

OSCOAP provides protection of CoAP payload, certain options, and header fields, as well as a secure binding between CoAP request and response messages, and freshness of requests and responses. It may be used in extremely constrained settings, where DTLS cannot be supported. Alternatively, OSCOAP can be combined with DTLS, thereby enabling end-to-end security of CoAP payload, in combination with hop-by-hop protection of the entire CoAP message, during transport between end-point and intermediary node. Examples of the use of OSCOAP are given in Appendix B.

The message protection provided by OSCOAP can alternatively be applied only to the payload of individual messages. We call this object security of content (OSCON) and it is defined in Appendix C.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. These words may also appear in this document in lowercase, absent their normative meanings.

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

Two different scopes of object security are defined:

- o OSCOAP = object security of CoAP, signaled with the Object-Security option.

- o OSCON = object security of content, signaled with Content Format/Media Type set to application/oscon (defined in Appendix C).

2. The Object-Security Option

The Object-Security option indicates that OSCOAP is used to protect the CoAP message exchange. The protection is achieved by means of a COSE object included in the protected CoAP message, as detailed in Section 5.

The Object-Security option is critical, safe to forward, part of the cache key, and not repeatable. Figure 2 illustrates the structure of the Object-Security option.

A CoAP proxy SHOULD NOT cache a response to a request with an Object-Security option, since the response is only applicable to the original client's request. The Object-Security option is included in the cache key for backward compatibility with proxies not recognizing the Object-Security option. The effect of this is that messages with the Object-Security option will never generate cache hits. To further prevent caching, a Max-Age option with value zero SHOULD be added to the protected CoAP responses.

No.	C	U	N	R	Name	Format	Length
TBD	x				Object-Security	opaque	0-

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Figure 2: The Object-Security Option

The length of the Object-Security option depends on whether the unprotected message has payload, on the set of options that are included in the unprotected message, the length of the integrity tag, and the length of the information identifying the security context.

- o If the unprotected message has payload, then the COSE object is the payload of the protected message (see Section 6.2 and Section 6.4), and the Object-Security option has length zero. An endpoint receiving a CoAP message with payload, that also contains a non-empty Object-Security option SHALL treat it as malformed and reject it.
- o If the unprotected message does not have payload, then the COSE object is the value of the Object-Security option and the length of the Object-Security option is equal to the size of the COSE object. An endpoint receiving a CoAP message without payload,

that also contains an empty Object-Security option SHALL treat it as malformed and reject it.

More details about the message overhead caused by the Object-Security option is given in Appendix A.

3. The Security Context

OSCOAP uses COSE with an Authenticated Encryption with Additional Data (AEAD) algorithm. The specification requires that client and server establish a security context to apply to the COSE objects protecting the CoAP messages. In this section we define the security context, and also specify how to establish a security context in client and server based on common shared secret material and a key derivation function (KDF).

The EDHOC protocol [I-D.selander-ace-cose-ecdhe] enables the establishment of secret material with the property of forward secrecy, and negotiation of KDF and AEAD, it thus provides all necessary pre-requisite steps for using OSCOAP as defined here.

3.1. Security Context Definition

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCOAP. Each security context is identified by a Context Identifier. A Context Identifier that is no longer in use can be reassigned to a new security context.

For each endpoint, the security context is composed by a "Common Context", a "Sender Context" and a "Recipient Context". The Common Context includes common security material. The endpoint protects the messages sent using the Sender Context. The endpoint verifies the messages received using the Recipient Context. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Note that, because of that, the two security contexts identified by the same Context Identifiers in the two endpoints are not the same, but they are partly mirrored.

An example is shown in Figure 3.

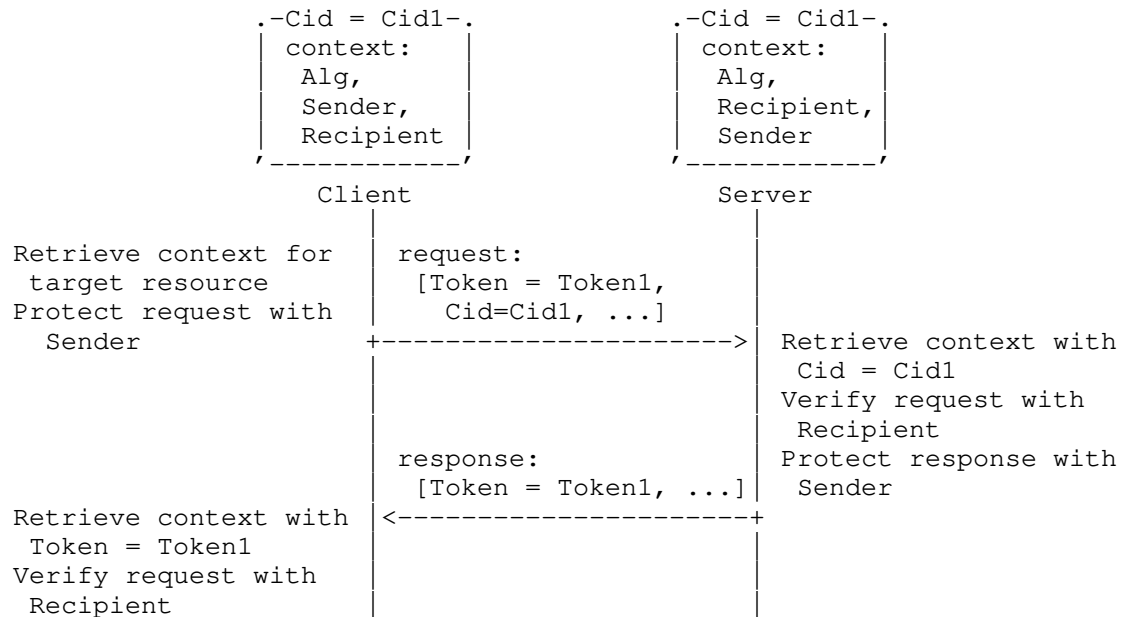


Figure 3: Retrieval and use of the Security Context

The Common Context structure contains the following parameters:

- o Context Identifier (Cid). Variable length byte string that identifies the security context. Its value is immutable once the security context is established.
- o Algorithm (Alg). Value that identifies the COSE AEAD algorithm to use for encryption. Its value is immutable once the security context is established.
- o Base Key (base_key). Byte string containing the key used to derive the security context Section 3.2.

The Sender Context structure contains the following parameters:

- o Sender ID. Variable length byte string identifying oneself. Its value is immutable once the security context is established.
- o Sender Key. Byte string containing the symmetric key to protect messages to send. Length is determined by Algorithm. Its value is immutable once the security context is established.
- o Sender IV. Byte string containing the fixed portion of IV (context IV in [I-D.ietf-cose-msg]) to protect messages to send.

Length is determined by Algorithm. Its value is immutable once the security context is established.

- o Sender Sequence Number. Non-negative integer enumerating the COSE objects that the endpoint sends, associated to the Context Identifier. It is used for replay protection, and to generate unique IVs for the AEAD. Maximum value is determined by Algorithm.

The Recipient Context structure contains the following parameters:

- o Recipient ID. Variable length byte string identifying the endpoint messages are received from or sent to. Its value is immutable once the security context is established.
- o Recipient Key. Byte string containing the symmetric key to verify messages received. Length is determined by the Algorithm. Its value is immutable once the security context is established.
- o Recipient IV. Byte string containing the context IV to verify messages received. Length is determined by Algorithm. Its value is immutable once the security context is established.
- o Recipient Sequence Number. Non-negative integer enumerating the COSE objects received, associated to the Context Identifier. It is used for replay protection, and to generate unique IVs for the AEAD. Maximum value is determined by Algorithm.
- o Replay Window. The replay protection window for messages received, equivalent to the functionality described in Section 4.1.2.6 of [RFC6347].

The 3-tuple (Cid, Sender ID, Sender Sequence Number) is called Transaction Identifier (Tid), and SHALL be unique for each COSE object and server. The Tid is used as a unique challenge in the COSE object of the protected CoAP request. The Tid is part of the Additional Authenticated Data (AAD, see Section 5) of the protected CoAP response message, which is how the challenge becomes signed by the server.

The client and server may change roles while maintaining the same security context. The former server will then make the request using the Sender Context, the former client will verify the request using its Recipient Context etc.

3.2. Security Context Establishment

This section aims at describing how to establish the security context, given some input parameters. The input parameters, which are established in a previous phase, are:

- o Context Identifier (Cid)
- o Algorithm (Alg)
- o Base Key (base_key)
- o Sender ID
- o Recipient ID
- o Replay Window (optionally)

These are included unchanged in the security context. We give below some indications on how applications should select these parameters. Moreover, the following parameters are established as described below:

- o Sender Key
- o Sender IV
- o Sender Sequence Number
- o Recipient Key
- o Recipient IV
- o Recipient Sequence Number
- o Replay Window

3.2.1. Derivation of Sender Key/IV, Recipient Key/IV

Given a common shared secret material and a common key derivation function, the client and server can derive the security context necessary to run OSCOAP. The derivation procedure described here MUST NOT be executed more than once on a set of common secret material. Also, the same base_key SHOULD NOT be used in different security contexts (identified by different Cids).

The procedure assumes that the common shared secret material is uniformly random and that the key derivation function is HKDF

[RFC5869]. This is for example the case after having used EDHOC [I-D.selander-ace-cose-ecdhe].

Assumptions:

- o The hash function, denoted HKDF, is the HMAC based key derivation function defined in [RFC5869] with specified hash function
- o The common shared secret material, denoted base_key, is uniformly pseudo-random of length at least equal to the output of the specified hash function

The security context parameters Sender Key/IV, Recipient Key/IV SHALL be derived using the HKDF-Expand primitive [RFC5869]:

output parameter = HKDF-Expand(base_key, info, key_length),

where:

- o base_key is defined above
- o info = Cid || Sender ID/Recipient ID || "IV"/"Key" || Algorithm || key_length
- o key_length is the key size of the AEAD algorithm

The Sender/Recipient Key shall be derived using the Cid concatenated with the Sender/Recipient ID, the label "Key", the Algorithm and the key_length. The Sender/Recipient IV shall be derived using the Cid concatenated with the Sender/Recipient ID, the label "IV", the Algorithm and the key_length.

For example, for the algorithm AES-CCM-64-64-128 (see Section 10.2 in [I-D.ietf-cose-msg]), key_length for the keys is 128 bits and key_length for the context IVs is 56 bits.

3.2.2. Sequence Numbers and Replay Window

The values of the Sequence Numbers are initialized to 0 during establishment of the security context. The default Replay Window size of 64 is used if no input parameter is provided in the set up phase.

3.2.3. Context Identifier and Sender/Recipient ID

As mentioned, Cid, Sender ID and Recipient ID are established in a previous phase. How this is done is application specific, but some guidelines are given in this section.

It is RECOMMENDED that the application uses 64-bits long pseudo-random Cids, in order to have globally unique Context Identifiers. Cid SHOULD be unique in the sets of all security contexts used by all the endpoints. If it is not the case, it is the role of the application to specify how to handle collisions.

In the same phase during which the Cid is established in the endpoint, the application informs the endpoint what resource can be accessed using the corresponding security context. The granularity of that is decided by the application (resource, host, etc). The endpoint SHALL save the association resource-Cid, in order to be able to retrieve the correct security context to access a resource.

The Sender ID and Recipient ID are also established in the endpoint during the previous set up phase. The application SHOULD make sure that these identifiers are locally unique in the set of all endpoints using the same security context. If it is not the case, it is again the role of the application to specify how to handle collisions.

In case of EDHOC [I-D.selander-ace-cose-ecdhe]) the Cid is the hash of the messages exchanged.

4. Protected CoAP Message Fields

This section defines how the CoAP message fields are protected. OSCOAP protects as much of the unprotected CoAP message as possible, while still allowing forward proxy operations [I-D.hartke-core-e2e-security-reqs].

The CoAP Payload SHALL be encrypted and integrity protected.

The CoAP Header fields Version and Code SHALL be integrity protected but not encrypted. The CoAP Message Layer parameters, Type and Message ID, as well as Token and Token Length SHALL neither be integrity protected nor encrypted.

Protection of CoAP Options can be summarized as follows:

- o To prevent information leakage, Uri-Path and Uri-Query SHALL be encrypted. As a consequence, if Proxy-Uri is used, those parts of the URI SHALL be removed from the Proxy-Uri. The CoAP Options Uri-Host, Uri-Port, Proxy-Uri, and Proxy-Scheme SHALL neither be encrypted, nor integrity protected (cf. protection of the effective request URI in Section 5.2).
- o The other CoAP options SHALL be encrypted and integrity protected.

A summary of which options are encrypted or integrity protected is shown in Figure 4.

No.	C	U	N	R	Name	Format	Length	E	D
1	x			x	If-Match	opaque	0-8	x	
3	x	x	-		Uri-Host	string	1-255		
4				x	ETag	opaque	1-8	x	
5	x				If-None-Match	empty	0	x	
6		x	-		Observe	uint	0-3	x	x
7	x	x	-		Uri-Port	uint	0-2		
8				x	Location-Path	string	0-255	x	
11	x	x	-	x	Uri-Path	string	0-255	x	
12					Content-Format	uint	0-2	x	
14		x	-		Max-Age	uint	0-4	x	x
15	x	x	-	x	Uri-Query	string	0-255	x	
17	x				Accept	uint	0-2	x	
20				x	Location-Query	string	0-255	x	
23	x	x	-	-	Block2	uint	0-3	x	x
27	x	x	-	-	Block1	uint	0-3	x	x
28			x		Size2	unit	0-4	x	x
35	x	x	-		Proxy-Uri	string	1-1034		
39	x	x	-		Proxy-Scheme	string	1-255		
60			x		Size1	uint	0-4	x	x

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt and Integrity Protect, D=Duplicate.

Figure 4: Protection of CoAP Options

Unless specified otherwise, CoAP options not listed in Figure 4 SHALL be encrypted and integrity protected.

The encrypted options are in general omitted from the protected CoAP message and not visible to intermediary nodes (see Section 6.2 and Section 6.4). Hence the actions resulting from the use of corresponding options is analogous to the case of communicating directly with the endpoint. For example, a client using an ETag option will not be served by a proxy.

However, some options which are encrypted need to be readable in the protected CoAP message to support certain proxy functions. A CoAP option which may be both encrypted in the COSE object of the protected CoAP message, and also unencrypted as CoAP option in the protected CoAP message, is called "duplicate". The "encrypted" value of a duplicate option is intended for the destination endpoint and

the "unencrypted" value is intended for a proxy. The unencrypted value is not integrity protected.

- o The Max-Age option is duplicate. The unencrypted Max-Age SHOULD have value zero to prevent caching of responses. The encrypted Max-Age is used as defined in [RFC7252] taking into account that it is not accessible to proxies.
- o The Observe option is duplicate. If Observe is used, then the encrypted Observe and the unencrypted Observe SHALL have the same value. The Observe option as used here targets the requirements on forwarding of [I-D.hartke-core-e2e-security-reqs] (Section 2.2.1.2).
- o The block options Block1 and Block2 are duplicate. The encrypted block options is used for end-to-end secure fragmentation of payload into blocks and protected information about the fragmentation (block number, last block, etc.). The MAC from each block is included in the calculation of the MAC for the next block's (see Section 5.2). In this way, each block in ordered sequence from the first block can be verified as it arrives. The unencrypted block option allows for arbitrary proxy fragmentation operations which cannot be verified by the endpoints. An intermediary node can generate an arbitrarily long sequence of blocks. However, since it is possible to protect fragmentation of large messages, there SHALL be a security policy defining a maximum unfragmented message size such that messages exceeding this size SHALL be fragmented by the sending endpoint. Hence an endpoint receiving fragments of a message that exceeds maximum message size SHALL discard this message.
- o The size options Size1 and Size2 are duplicate, analogously to the block options.

Specifications of new CoAP options SHOULD specify how they are processed with OSCOAP. New CoAP options SHALL be encrypted and integrity protected. New CoAP options SHOULD NOT be duplicate unless a forwarding proxy needs to read the option. If an option is registered as duplicate, the duplicate value SHOULD NOT be the same as the end-to-end value, unless the proxy is required by specification to be able to read the end-to-end value.

5. The COSE Object

This section defines how to use the COSE format [I-D.ietf-cose-msg] to wrap and protect data in the unprotected CoAP message. OSCOAP uses the COSE_Encrypt0 structure with an Authenticated Encryption with Additional Data (AEAD) algorithm.

The mandatory to support AEAD algorithm is AES-CCM-64-64-128 defined in Section 10.2 of [I-D.ietf-cose-msg]. For AES-CCM-64-64-128 the length of Sender Key and Recipient Key SHALL be 128 bits, the length of IV, Sender IV, and Recipient IV SHALL be 7 bytes, and the maximum Sender Sequence Number and Recipient Sequence Number SHALL be $2^{56}-1$. The IV is constructed using a Partial IV exactly like in Section 3.1 of [I-D.ietf-cose-msg], i.e. by padding the Sender Sequence Number or the Recipient Sequence Number with zeroes and XORing it with the Sender IV or Recipient IV, respectively.

Since OSCOAP only makes use of a single COSE structure, there is no need to explicitly specify the structure, and OSCOAP uses the untagged version of the COSE_Encrypt0 structure (Section 2. of [I-D.ietf-cose-msg]). If the COSE object has a different structure, the recipient MUST reject the message, treating it as malformed.

We denote by Plaintext the data that is encrypted and integrity protected, and by Additional Authenticated Data (AAD) the data that is integrity protected only, in the COSE object.

The fields of COSE_Encrypt0 structure are defined as follows (see example in Appendix C.4).

- o The "Headers" field is formed by:
 - * The "protected" field, which SHALL include:
 - + The "Partial IV" parameter. The value is set to the Sender Sequence Number. The Partial IV is a byte string (type: bstr), where the length is the minimum length needed to encode the sequence number. An Endpoint that receives a COSE object with a sequence number encoded with leading zeroes (i.e. longer than the minimum needed length) SHALL reject the corresponding message as malformed.
 - + If the message is a CoAP request, the "kid" parameter. The value is set to the Context Identifier (see Section 3).
 - + Optionally, the parameter called "sid", defined below. The value is set to the Sender ID (see Section 3). Note that since this parameter is sent in clear, privacy issues SHOULD be considered by the application defining the Sender ID.
 - * The "unprotected" field, which SHALL be empty.
- o The "cipher text" field is computed from the Plaintext (see Section 5.1) and the Additional Authenticated Data (AAD) (see

Section 5.2) and encoded as a byte string (type: bstr), following Section 5.2 of [I-D.ietf-cose-msg].

sid: This parameter is used to identify the sender of the message. Applications MUST NOT assume that 'sid' values are unique. This is not a security critical field. For this reason, it can be placed in the unprotected headers bucket.

name	label	value type	value registry	description
sid	TBD	bstr		Sender identifier

Table 1: Additional COSE Header Parameter

5.1. Plaintext

The Plaintext is formatted as a CoAP message without Header (see Figure 5) consisting of:

- o all CoAP Options present in the unprotected message which are encrypted (see Section 4), in the order as given by the Option number (each Option with Option Header including delta to previous included encrypted option); and
- o the CoAP Payload, if present, and in that case prefixed by the one-byte Payload Marker (0xFF).

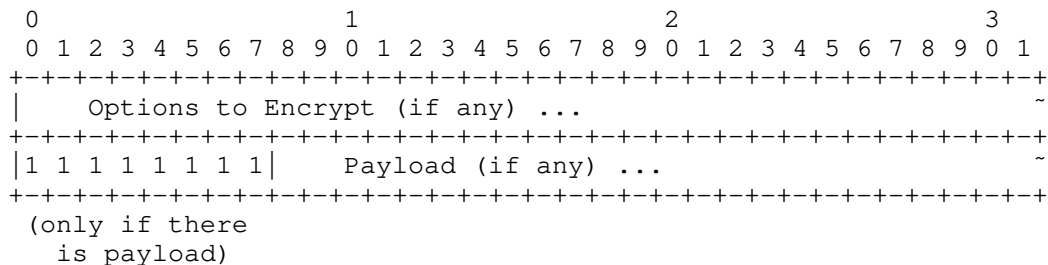


Figure 5: Plaintext

5.2. Additional Authenticated Data

The Additional Authenticated Data ("Enc_structure") as described in Section 5.3 of [I-D.ietf-cose-msg] includes:

- o the "context" parameter, which has value "Encrypted"

- o the "protected" parameter, which includes the "protected" part of the "Headers" field;
- o the "external_aad" is a serialized CBOR array (see Figure 8) that contains, in the given order:
 - * ver: uint, contains the CoAP version number of the unprotected CoAP message, as defined in Section 3 of [RFC7252]
 - * code: bstr, contains is the CoAP Code of the unprotected CoAP message, as defined in Section 3 of [RFC7252].
 - * alg: bstr, contains the serialized Algorithm from the security context used for the exchange (see Section 3.1);
 - * request-uri: tstr, contains the plaintext "effective" request URI composed from the request scheme and Uri-* options according to the method described in Section 6.5 of [RFC7252], if the message is a CoAP request;
 - * transaction-id: bstr, only included if the message to protect or verify is a CoAP response, contains the Transaction Identifier (Tid) of the associated CoAP request (see Section 3). Note that the Tid is the 3-tuple (Cid, Sender ID, Sender Sequence Number) for the endpoint sending the request and verifying the response; which means that for the endpoint sending the response, the Tid has value (Cid, Recipient ID, seq), where seq is the value of the "Partial IV" in the COSE object of the request (see Section 5); and
 - * mac-previous-block: bstr, contains the MAC of the message containing the previous block in the sequence, as enumerated by Block1 in the case of a request and Block2 in the case of a response, if the message is fragmented using a block option [RFC7959].

```
external_aad_req = [  
    ver : uint,  
    code : bstr,  
    alg : bstr,  
    request-uri : tstr,  
    ? mac-previous-block : bstr  
]
```

Figure 6: external_aad for a request

```
external_aad_resp = [  
    ver : uint,  
    code : bstr,  
    alg : bstr,  
    transaction-id : bstr,  
    ? mac-previous-block : bstr  
]
```

Figure 7: external_aad for a response

```
external_aad = external_aad_req / external_aad_resp
```

Figure 8: external_aad

The encryption process is described in Section 5.3 of [I-D.ietf-cose-msg].

6. Protecting CoAP Messages

6.1. Replay and Freshness Protection

In order to protect from replay of messages and verify freshness, a CoAP endpoint SHALL maintain a Sender Sequence Number, and a Recipient Sequence Number associated to a security context, which is identified with a Context Identifier (Cid). The two sequence numbers are the highest sequence number the endpoint has sent and the highest sequence number the endpoint has received. An endpoint uses the Sender Sequence Number to protect messages to send and the Recipient Sequence Number to verify received messages, as described in Section 3.

Depending on use case and ordering of messages provided by underlying layers, an endpoint MAY maintain a sliding replay window for Sequence Numbers of received messages associated to each Cid. In case of reliable transport, the receiving endpoint MAY require that the Sequence Number of a received message equals last Sequence Number + 1.

A receiving endpoint SHALL verify that the Sequence Number received in the COSE object has not been received before in the security context identified by the Cid. The receiving endpoint SHALL also reject messages with a sequence number greater than $2^{56}-1$.

OSCOAP is a challenge-response protocol, where the response is verified to match a prior request, by including the unique transaction identifier (Tid as defined in Section 3) of the request in the Additional Authenticated Data of the response message.

If a CoAP server receives a request with the Object-Security option, then the server SHALL include the Tid of the request in the AAD of the response, as described in Section 6.4.

If the CoAP client receives a response with the Object-Security option, then the client SHALL verify the integrity of the response, using the Tid of its own associated request in the AAD, as described in Section 6.5.

6.2. Protecting the Request

Given an unprotected CoAP request, including header, options and payload, the client SHALL perform the following steps to create a protected CoAP request using a security context associated with the target resource (see Section 3.2.3).

1. Increment the Sender Sequence Number by one (note that this means that sequence number 0 is never used). If the Sender Sequence Number exceeds the maximum number for the AEAD algorithm, the client MUST NOT process any requests with the given security context. The client SHOULD acquire a new security context (and consequently inform the server about it) before this happens. The latter is out of scope of this memo.
2. Compute the COSE object as specified in Section 5
 - * the IV in the AEAD is created by XORing the Sender IV (context IV) with the Sender Sequence Number (partial IV).
 - * If the block option is used, the AAD includes the MAC from the previous fragment sent (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-sent fragment to compute the following.
 - * Note that the 'sid' field containing the Sender ID is included in the COSE object (Section 5) if the application needs it.
3. Format the protected CoAP message as an ordinary CoAP message, with the following Header, Options, and Payload, based on the unprotected CoAP message:
 - * The CoAP header is the same as the unprotected CoAP message.
 - * The CoAP options which are encrypted and not duplicate (Section 4) are removed. Any duplicate option which is present has its unencrypted value. The Object-Security option is added.

- * If the message type of the unprotected CoAP message does not allow Payload, then the value of the Object-Security option is the COSE object. If the message type of the unprotected CoAP message allows Payload, then the Object-Security option is empty and the Payload of the protected CoAP message is the COSE object.
4. Store in memory the association Token - Cid. The Client SHALL be able to find the correct security context used to protect the request and verify the response with use of the Token of the message exchange.

6.3. Verifying the Request

A CoAP server receiving a message containing the Object-Security option SHALL perform the following steps, using the security context identified by the Context Identifier in the "kid" parameter in the received COSE object:

1. Verify the Sequence Number in the Partial IV parameter, as described in Section 6.1. If it cannot be verified that the Sequence Number has not been received before, the server MUST stop processing the request.
2. Recreate the Additional Authenticated Data, as described in Section 5.
 - * If the block option is used, the AAD includes the MAC from the previous fragment received (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-received fragment to compute the following.
3. Compose the IV by XORing the Recipient IV (context IV) with the Partial IV parameter, received in the COSE Object.
4. Retrieve the Recipient Key.
5. Verify and decrypt the message. If the verification fails, the server MUST stop processing the request.
6. If the message verifies, update the Recipient Sequence Number or Replay Window, as described in Section 6.1.
7. Restore the unprotected request by adding any decrypted options or payload from the plaintext. Any duplicate options (Section 4) are overwritten. The Object-Security option is removed.

6.4. Protecting the Response

A server receiving a valid request with a protected CoAP message (i.e. containing an Object-Security option) SHALL respond with a protected CoAP message.

Given an unprotected CoAP response, including header, options, and payload, the server SHALL perform the following steps to create a protected CoAP response, using the security context identified by the Context Identifier of the received request:

1. Increment the Sender Sequence Number by one (note that this means that sequence number 0 is never used). If the Sender Sequence Number exceeds the maximum number for the AEAD algorithm, the server MUST NOT process any more responses with the given security context. The server SHOULD acquire a new security context (and consequently inform the client about it) before this happens. The latter is out of scope of this memo.
2. Compute the COSE object as specified in Section 5
 - * The IV in the AEAD is created by XORing the Sender IV (context IV) and the Sender Sequence Number.
 - * If the block option is used, the AAD includes the MAC from the previous fragment sent (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-sent fragment to compute the following.
3. Format the protected CoAP message as an ordinary CoAP message, with the following Header, Options, and Payload based on the unprotected CoAP message:
 - * The CoAP header is the same as the unprotected CoAP message.
 - * The CoAP options which are encrypted and not duplicate (Section 4) are removed. Any duplicate option which is present has its unencrypted value. The Object-Security option is added.
 - * If the message type of the unprotected CoAP message does not allow Payload, then the value of the Object-Security option is the COSE object. If the message type of the unprotected CoAP message allows Payload, then the Object-Security option is empty and the Payload of the protected CoAP message is the COSE object.

Note the differences between generating a protected request, and a protected response, for example whether "kid" is present in the header, or whether Destination URI or Tid is present in the AAD, of the COSE object.

6.5. Verifying the Response

A CoAP client receiving a message containing the Object-Security option SHALL perform the following steps, using the security context identified by the Token of the received response:

1. Verify the Sequence Number in the Partial IV parameter as described in Section 6.1. If it cannot be verified that the Sequence Number has not been received before, the client MUST stop processing the response.
2. Recreate the Additional Authenticated Data as described in Section 5.
 - * If the block option is used, the AAD includes the MAC from the previous fragment received (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-received fragment to compute the following.
3. Compose the IV by XORing the Recipient IV (context IV) with the Partial IV parameter, received in the COSE Object.
4. Retrieve the Recipient Key.
5. Verify and decrypt the message. If the verification fails, the client MUST stop processing the response.
6. If the message verifies, update the Recipient Sequence Number or Replay Window, as described in Section 6.1.
7. Restore the unprotected response by adding any decrypted options or payload from the plaintext. Any duplicate options (Section 4) are overwritten. The Object-Security option is removed.

7. Security Considerations

In scenarios with intermediary nodes such as proxies or brokers, transport layer security such as DTLS only protects data hop-by-hop. As a consequence the intermediary nodes can read and modify information. The trust model where all intermediate nodes are considered trustworthy is problematic, not only from a privacy perspective, but also from a security perspective, as the

intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases, where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture brittle.

DTLS protects hop-by-hop the entire CoAP message, including header, options, and payload. OSCOAP protects end-to-end the payload, and all information in the options and header, that is not required for forwarding (see Section 4). DTLS and OSCOAP can be combined, thereby enabling end-to-end security of CoAP payload, in combination with hop-by-hop protection of the entire CoAP message, during transport between end-point and intermediary node.

The CoAP message layer, however, cannot be protected end-to-end through intermediary devices since the parameters Type and Message ID, as well as Token and Token Length may be changed by a proxy. Moreover, messages that are not possible to verify should for security reasons not always be acknowledged but in some cases be silently dropped. This would not comply with CoAP message layer, but does not have an impact on the application layer security solution, since message layer is excluded from that.

The use of COSE to protect CoAP messages as specified in this document requires an established security context. The method to establish the security context described in Section 3.2 is based on a common shared secret material and key derivation function in client and server. EDHOC [I-D.selander-ace-cose-ecdhe] describes an augmented Diffie-Hellman key exchange to produce forward secret keying material and agree on crypto algorithms necessary for OSCOAP, authenticated with pre-established credentials. These pre-established credentials may, in turn, be provisioned using a trusted third party such as described in the OAuth-based ACE framework [I-D.ietf-ace-oauth-authz]. An OSCOAP profile of ACE is described in [I-D.seitz-ace-oscoap-profile].

For symmetric encryption it is required to have a unique IV for each message, for which the sequence numbers in the COSE message field "Partial IV" is used. The context IVs (Sender IV and Recipient IV) SHOULD be established between sender and recipient before the message is sent, for example using the method in [I-D.selander-ace-cose-ecdhe], to avoid the overhead of sending it in each message.

The MTI AEAD algorithm AES-CCM-64-64-128 is selected for broad applicability in terms of message size (2^{64} blocks) and maximum no. messages ($2^{56}-1$). For 128 bit CCM*, use instead AES-CCM-16-64-128 [I-D.ietf-cose-msg].

If the recipient accepts any sequence number larger than the one previously received (less than the maximum sequence number), then the problem of sequence number synchronization is avoided. With reliable transport it may be defined that only messages with sequence number which are equal to previous sequence number + 1 are accepted. The alternatives to sequence numbers have their issues: very constrained devices may not be able to support accurate time, or to generate and store large numbers of random IVs. The requirement to change key at counter wrap is a complication, but it also forces the user of this specification to think about implementing key renewal.

The encrypted block options enable the sender to split large messages into protected fragments such that the receiving node can verify blocks before having received the complete message. In order to protect from attacks replacing fragments from a different message with the same block number between same endpoints and same resource at roughly the same time, the MAC from the message containing one block is included in the external_aad of the message containing the next block.

The unencrypted block options allow for arbitrary proxy fragmentation operations which cannot be verified by the endpoints, but can by policy be restricted in size since the encrypted options allow for secure fragmentation of very large messages. A maximum message size (above which the sending endpoint fragments the message and the receiving endpoint discards the message, if complying to the policy) may be obtained as part of normal resource discovery.

8. Privacy Considerations

Privacy threats executed through intermediate nodes are considerably reduced by means of OSCOAP. End-to-end integrity protection and encryption of CoAP payload and all options that are not used for forwarding, provide mitigation against attacks on sensor and actuator communication, which may have a direct impact on the personal sphere.

CoAP headers sent in plaintext allow for example matching of CON and ACK (CoAP Message Identifier), matching of request and responses (Token) and traffic analysis.

9. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[[this document]]" with the RFC number of this specification.

9.1. Sid Registration

IANA is requested to enter a new parameter entitled "sid" to the registry "COSE Header Parameters". The parameter is defined in Table 1.

9.2. CoAP Option Number Registration

The Object-Security option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	Object-Security	[[this document]]

9.3. Media Type Registrations

The "application/oscon" media type is added to the Media Types registry:

Type name: application

Subtype name: cose

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section of [[this document]].

Interoperability considerations: N/A

Published specification: [[this document]]

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

- * Magic number(s): N/A

- * File extension(s): N/A

- * Macintosh file type code(s): N/A

Person & email address to contact for further information:
iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander, goran.selander@ericsson.com

Change Controller: IESG

Provisional registration? No

9.4. CoAP Content Format Registration

The "application/oscon" content format is added to the CoAP Content Format registry:

Media type	Encoding	ID	Reference
application/oscon	-	70	[[this document]]

10. Acknowledgments

Klaus Hartke has independently been working on the same problem and a similar solution: establishing end-to-end security across proxies by adding a CoAP option. We are grateful to Malisa Vucinic and Marco Tiloca for providing helpful and timely reviews of previous versions of the draft. We are also grateful to Carsten Bormann and Jim Schaad for providing input and interesting discussions.

11. References

11.1. Normative References

- [I-D.ietf-cose-msg] Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-20 (work in progress), October 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<http://www.rfc-editor.org/info/rfc7959>>.

11.2. Informative References

- [I-D.bormann-6lo-coap-802-15-ie]
Bormann, C., "Constrained Application Protocol (CoAP) over IEEE 802.15.4 Information Element for IETF", draft-bormann-6lo-coap-802-15-ie-00 (work in progress), April 2016.
- [I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-01 (work in progress), July 2016.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-02 (work in progress), June 2016.
- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-04 (work in progress), August 2016.
- [I-D.seitz-ace-oscoap-profile]
Seitz, L., "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-00 (work in progress), July 2016.
- [I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-02 (work in progress), July 2016.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<http://www.rfc-editor.org/info/rfc5869>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.

Appendix A. Overhead

OSCOAP transforms an unprotected CoAP message to a protected CoAP message, and the protected CoAP message is larger than the unprotected CoAP message. This appendix illustrates the message expansion.

A.1. Length of the Object-Security Option

The protected CoAP message contains the COSE object. The COSE object is included in the payload if the message type of the unprotected CoAP message allows payload or else in the Object-Security option. In the former case the Object-Security option is empty. So the length of the Object-Security option is either zero or the size of the COSE object, depending on whether the CoAP message allows payload or not.

Length of Object-Security option = { 0, size of COSE Object }

A.2. Size of the COSE Object

The size of the COSE object is the sum of the sizes of

- o the Header parameters,
- o the Cipher Text (excluding the Tag),
- o the Tag, and
- o data incurred by the COSE format itself (including CBOR encoding).

Let's analyse the contributions one at a time:

- o The header parameters of the COSE object are the Context Identifier (Cid) and the Sequence Number (Seq) (also known as the Transaction Identifier (Tid)) if the message is a request, and Seq only if the message is a response (see Section 5).
 - * The size of Cid depends on the number of simultaneous clients, as discussed in Section 3.2
 - * The size of Seq is variable, and increases with the number of messages exchanged.
 - * As the IV is generated from the padded Sequence Number and a previously agreed upon context IV it is not required to send the whole IV in the message.

- o The Cipher Text, excluding the Tag, is the encryption of the payload and the encrypted options Section 4, which are present in the unprotected CoAP message.
- o The size of the Tag depends on the Algorithm. For example, for the algorithm AES-CCM-64-64-128, the Tag is 8 bytes.
- o The overhead from the COSE format itself depends on the sizes of the previous fields, and is of the order of 10 bytes.

A.3. Message Expansion

The message expansion is not the size of the COSE object. The cipher text in the COSE object is encrypted payload and options of the unprotected CoAP message – the plaintext of which is removed from the protected CoAP message. Since the size of the cipher text is the same as the corresponding plaintext, there is no message expansion due to encryption; payload and options are just represented in a different way in the protected CoAP message:

- o The encrypted payload is in the payload of the protected CoAP message
- o The encrypted options are in the Object-Security option or within the payload.

Therefore the OSCOAP message expansion is due to Cid (if present), Seq, Tag, and COSE overhead:

$$\text{Message Overhead} = \text{Cid} + \text{Seq} + \text{Tag} + \text{COSE Overhead}$$

Figure 9: OSCOAP message expansion

A.4. Example

This section gives an example of message expansion in a request with OSCOAP.

In this example we assume an extreme 4-byte Cid, based on the assumption of an ACE deployment with billions of clients requesting access to this particular server. (A typical Cid, will be 1-2 byte as is discussed in Appendix A.2.)

- o Cid: 0xa1534e3c

In the example the sequence number is 225, requiring 1 byte to encode. (The size of Seq could be larger depending on how many messages that has been sent as is discussed in Appendix A.2.)

- o Seq: 225

The example is based on AES-CCM-64-64-128.

- o Tag is 8 bytes

The COSE object is represented in Figure 10 using CBOR's diagnostic notation.

```
[
  h'a20444a1534e3c0641e2', # protected:
                             {04:h'a1534e3c',
                             06:h'e2'}
  {},                        # unprotected: -
  Tag                        # cipher text + 8 byte authentication tag
]
```

Figure 10: Example of message expansion

Note that the encrypted CoAP options and payload are omitted since we target the message expansion (see Appendix A.3). Therefore the size of the COSE Cipher Text equals the size of the Tag, which is 8 bytes.

The COSE object encodes to a total size of 22 bytes, which is the message expansion in this example. The COSE overhead in this example is $22 - (4 + 1 + 8) = 9$ bytes, according to the formula in Figure 9. Note that in this example two bytes in the COSE overhead are used to encode the length of Cid and the length of Seq.

Figure 11 summarizes these results.

Tid	Tag	COSE OH	Message OH
5 bytes	8 bytes	9 bytes	22 bytes

Figure 11: Message overhead for a 5-byte Tid and 8-byte Tag.

Appendix B. Examples

This section gives examples of OSCOAP. The message exchanges are made, based on the assumption that there is a security context established between client and server. For simplicity, these examples only indicate the content of the messages without going into detail of the COSE message format.

B.1. Secure Access to Sensor

Here is an example targeting the scenario in the Section 2.2.1. - Forwarding of [I-D.hartke-core-e2e-security-reqs]. The example illustrates a client requesting the alarm status from a server. In the request, CoAP option Uri-Path is encrypted and integrity protected, and the CoAP header fields Code and Version are integrity protected (see Section 4). In the response, the CoAP Payload is encrypted and integrity protected, and the CoAP header fields Code and Version are integrity protected.

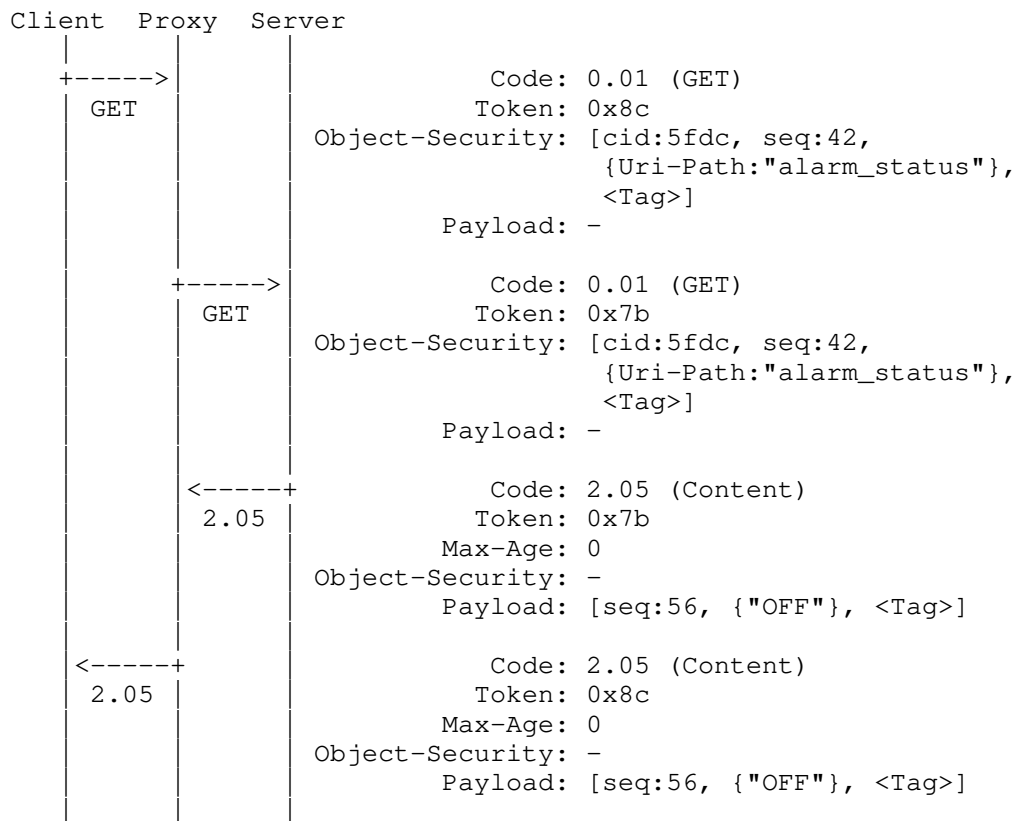


Figure 12: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicate a COSE object. The brackets { ... } indicate encrypted data.

Since the unprotected request message (GET) has no payload, the Object-Security option carries the COSE object as its value. Since the unprotected response message (Content) has payload ("OFF"), the COSE object (indicated with [...]) is carried as the CoAP payload.

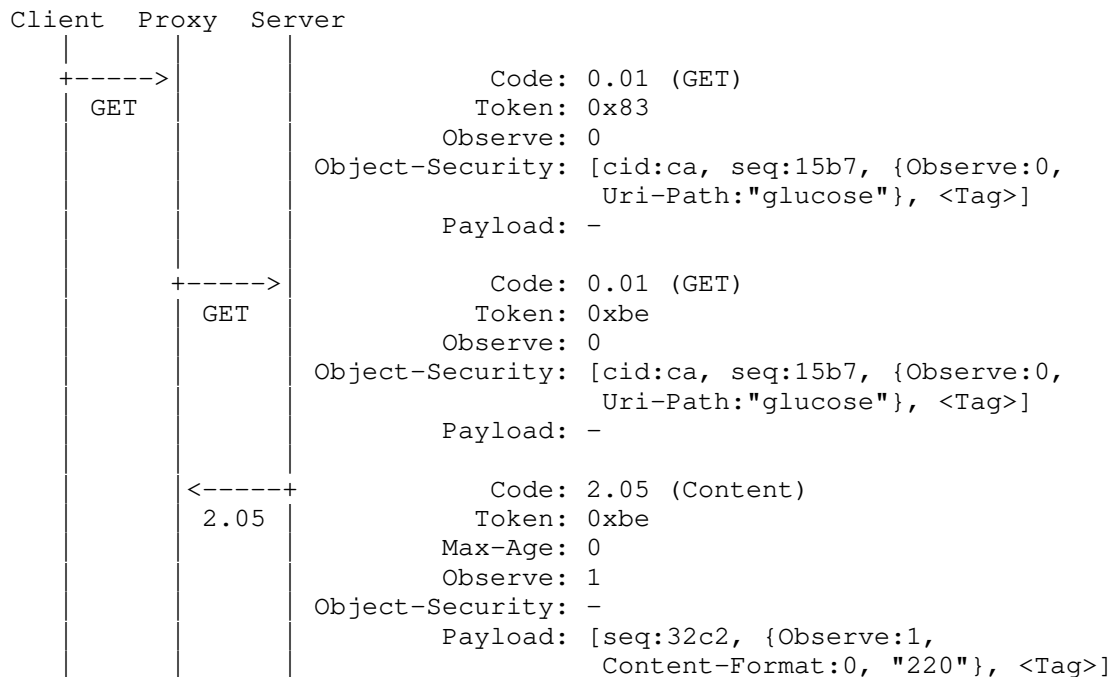
The COSE header of the request contains a Context Identifier (cid:5fdc), indicating which security context was used to protect the message and a Sequence Number (seq:42).

The option Uri-Path (alarm_status) and payload ("OFF") are formatted as indicated in Section 5, and encrypted in the COSE Cipher Text (indicated with { ... }).

The server verifies that the Sequence Number has not been received before (see Section 6.1). The client verifies that the Sequence Number has not been received before and that the response message is generated as a response to the sent request message (see Section 6.1).

B.2. Secure Subscribe to Sensor

Here is an example targeting the scenario in the Forwarding with observe case of [I-D.hartke-core-e2e-security-reqs]. The example illustrates a client requesting subscription to a blood sugar measurement resource (GET /glucose), and first receiving the value 220 mg/dl, and then a second reading with value 180 mg/dl. The CoAP options Observe, Uri-Path, Content-Format, and Payload are encrypted and integrity protected, and the CoAP header field Code is integrity protected (see Section 4).



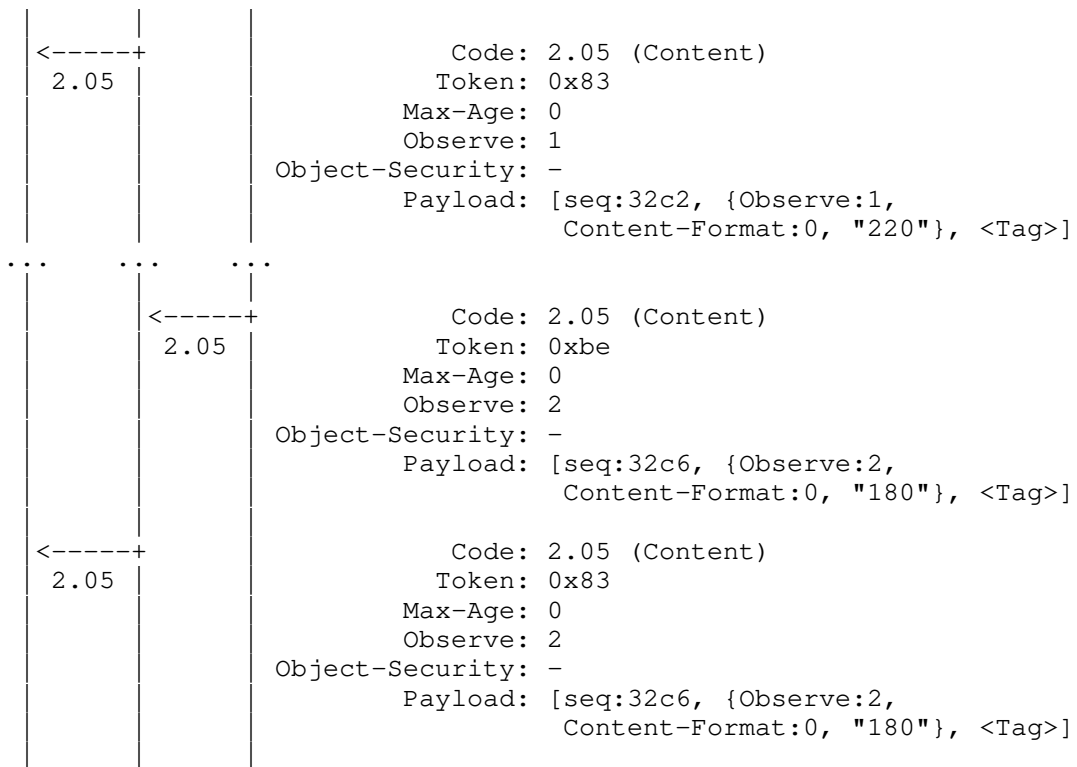


Figure 13: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicates COSE object. The bracket { ... } indicates encrypted data.

Since the unprotected request message (GET) allows no payload, the COSE object (indicated with [...]) is carried in the Object-Security option value. Since the unprotected response message (Content) has payload, the Object-Security option is empty, and the COSE object is carried as the payload.

The COSE header of the request contains a Context Identifier (cid:ca), indicating which security context was used to protect the message and a Sequence Number (seq:15b7).

The options Observe, Content-Format and the payload are formatted as indicated in Section 5, and encrypted in the COSE cipher text (indicated with { ... }).

The server verifies that the Sequence Number has not been received before (see Section 6.1). The client verifies that the Sequence

Number has not been received before and that the response message is generated as a response to the subscribe request.

Appendix C. Object Security of Content (OSCON)

OSCOAP protects message exchanges end-to-end between a certain client and a certain server, targeting the security requirements for forward proxy of [I-D.hartke-core-e2e-security-reqs]. In contrast, many use cases require one and the same message to be protected for, and verified by, multiple endpoints, see caching proxy section of [I-D.hartke-core-e2e-security-reqs]. Those security requirements can be addressed by protecting essentially the payload/content of individual messages using the COSE format ([I-D.ietf-cose-msg]), rather than the entire request/response message exchange. This is referred to as Object Security of Content (OSCON).

OSCON transforms an unprotected CoAP message into a protected CoAP message in the following way: the payload of the unprotected CoAP message is wrapped by a COSE object, which replaces the payload of the unprotected CoAP message. We call the result the "protected" CoAP message.

The unprotected payload shall be the plaintext/payload of the COSE object. The 'protected' field of the COSE object 'Headers' shall include the context identifier, both for requests and responses. If the unprotected CoAP message includes a Content-Format option, then the COSE object shall include a protected 'content type' field, whose value is set to the unprotected message Content-Format value. The Content-Format option of the protected CoAP message shall be replaced with "application/oscon" (Section 9)

The COSE object shall be protected (encrypted) and verified (decrypted) as described in ([I-D.ietf-cose-msg]).

In the case of symmetric encryption, the same key and IV shall not be used twice. Sequence numbers for partial IV as specified for OSCOAP may be used for replay protection as described in Section 6.1. The use of time stamps in the COSE header parameter 'operation time' [I-D.ietf-cose-msg] for freshness may be used.

OSCON shall not be used in cases where CoAP header fields (such as Code or Version) or CoAP options need to be integrity protected or encrypted. OSCON shall not be used in cases which require a secure binding between request and response.

The scenarios in Sections 3.3 - 3.5 of [I-D.hartke-core-e2e-security-reqs] assume multiple recipients for a particular content. In this case the use of symmetric keys does not

provide data origin authentication. Therefore the COSE object should in general be protected with a digital signature.

C.1. Overhead OSCON

In general there are four different kinds of ciphersuites that need to be supported: message authentication code, digital signature, authenticated encryption, and symmetric encryption + digital signature. The use of digital signature is necessary for applications with many legitimate recipients of a given message, and where data origin authentication is required.

To distinguish between these different cases, the tagged structures of COSE are used (see Section 2 of [I-D.ietf-cose-msg]).

The size of the COSE message for selected algorithms are detailed in this section.

The size of the header is shown separately from the size of the MAC/signature. A 4-byte Context Identifier and a 1-byte Sequence Number are used throughout all examples, with these values:

- o Cid: 0xa1534e3c
- o Seq: 0xa3

For each scheme, we indicate the fixed length of these two parameters ("Cid+Seq" column) and of the Tag ("MAC"/"SIG"/"TAG"). The "Message OH" column shows the total expansions of the CoAP message size, while the "COSE OH" column is calculated from the previous columns following the formula in Figure 9.

Overhead incurring from CBOR encoding is also included in the COSE overhead count.

To make it easier to read, COSE objects are represented using CBOR's diagnostic notation rather than a binary dump.

C.2. MAC Only

This example is based on HMAC-SHA256, with truncation to 8 bytes (HMAC 256/64).

Since the key is implicitly known by the recipient, the COSE_Mac0_Tagged structure is used (Section 6.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```

996(                                     # COSE_Mac0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                06:h'a3'}
    {},                         # unprotected
    h'',                        # payload
    MAC                         # truncated 8-byte MAC
  ]
)

```

This COSE object encodes to a total size of 26 bytes.

Figure 14 summarizes these results.

Structure	Tid	MAC	COSE OH	Message OH
COSE_Mac0_Tagged	5 B	8 B	13 B	26 B

Figure 14: Message overhead for a 5-byte Tid using HMAC 256/64

C.3. Signature Only

This example is based on ECDSA, with a signature of 64 bytes.

Since only one signature is used, the COSE_Sign1_Tagged structure is used (Section 4.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```

997(                                     # COSE_Sign1_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                06:h'a3'}
    {},                         # unprotected
    h'',                        # payload
    SIG                         # 64-byte signature
  ]
)

```

This COSE object encodes to a total size of 83 bytes.

Figure 15 summarizes these results.

Structure	Tid	SIG	COSE OH	Message OH
COSE_Sign1_Tagged	5 B	64 B	14 B	83 bytes

Figure 15: Message overhead for a 5-byte Tid using 64 byte ECDSA signature.

C.4. Authenticated Encryption with Additional Data (AEAD)

This example is based on AES-CCM with the MAC truncated to 8 bytes.

It is assumed that the IV is generated from the Sequence Number and some previously agreed upon context IV. This means it is not required to explicitly send the whole IV in the message.

Since the key is implicitly known by the recipient, the COSE_Encrypt0_Tagged structure is used (Section 5.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```

993(                                     # COSE_Encrypt0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                06:h'a3'}
    {},                         # unprotected
    TAG                         # cipher text + truncated 8-byte TAG
  ]
)

```

This COSE object encodes to a total size of 25 bytes.

Figure 16 summarizes these results.

Structure	Tid	TAG	COSE OH	Message OH
COSE_Encrypt0_Tagged	5 B	8 B	12 B	25 bytes

Figure 16: Message overhead for a 5-byte Tid using AES_128_CCM_8.

C.5. Symmetric Encryption with Asymmetric Signature (SEAS)

This example is based on AES-CCM and ECDSA with 64 bytes signature. The same assumption on the security context as in Appendix C.4. COSE defines the field 'counter signature w/o headers' that is used here to sign a COSE_Encrypt0_Tagged message (see Section 3 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```

993(                                     # COSE_Encrypt0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                06:h'a3'}
    {9:SIG},                     # unprotected:
                                09: 64 bytes signature
    TAG                          # cipher text + truncated 8-byte TAG
  ]
)

```

This COSE object encodes to a total size of 92 bytes.

Figure 17 summarizes these results.

Structure	Tid	TAG	SIG	COSE OH	Message OH
COSE_Encrypt0_Tagged	5 B	8 B	64 B	15 B	92 B

Figure 17: Message overhead for a 5-byte Tid using AES-CCM countersigned with ECDSA.

Authors' Addresses

Goeran Selander
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Ludwig Seitz
SICS Swedish ICT
Scheelevagen 17
Lund 22370
Sweden

Email: ludwig@sics.se

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: June 23, 2016

B. Silverajan
Tampere University of Technology
T. Savolainen
Nokia
December 21, 2015

CoAP Communication with Alternative Transports
draft-silverajan-core-coap-alternative-transports-09

Abstract

CoAP has been standardised as an application level REST-based protocol. A single CoAP message is typically encapsulated and transmitted using UDP or DTLS as transports. These transports are optimal solutions for CoAP use in IP-based constrained environments and nodes. However compelling motivation exists for allowing CoAP to operate with other transports and protocols. Examples are M2M communication in cellular networks using SMS, more suitable transport protocols for firewall/NAT traversal, end-to-end reliability and security such as TCP and TLS, or employing proxying and tunneling gateway techniques such as the WebSocket protocol. This draft examines the requirements for conveying CoAP messages to end points over such alternative transports. It also provides a new URI format for representing CoAP resources over alternative transports.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 23, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Usage Cases	4
2.1. Use of SMS	4
2.2. Use of WebSockets	4
2.3. Use of P2P Overlays	4
2.4. Use of TCP and TLS	5
3. Node Types based on Transport Availability	5
4. CoAP Alternative Transport URI	6
4.1. Design Considerations	7
4.2. URI format	8
5. Alternative Transport Analysis and Properties	9
6. IANA Considerations	11
7. Security Considerations	11
8. Acknowledgements	12
9. References	12
9.1. Normative References	12
9.2. Informative References	12
Appendix A. Expressing transport in the URI in other ways . . .	15
A.1. Transport information as part of the URI authority . . .	15
A.1.1. Usage of DNS records	16
A.2. Making CoAP Resources Available over Multiple Transports	16
A.3. Transport as part of a 'service:' URL scheme	18
Authors' Addresses	19

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] has been standardised by the CoRE WG as a lightweight, HTTP-like protocol providing a request/response model that constrained nodes can use to communicate with other nodes, be those servers, proxies, gateways, less constrained nodes, or other constrained nodes. CoAP has been defined to utilise UDP and DTLS as transports.

As the Internet evolves by integrating new kinds of networks, services and devices, the need for a consistent, lightweight method for resource representation, retrieval and manipulation becomes

evident. Owing to its simplicity and low overhead, CoAP is a highly suitable protocol for this purpose. However, communicating CoAP endpoints can reside in networks where end-to-end UDP-based communication can be challenging. These include networks separated by NATs and firewalls, cellular networks in which the Short Messaging Service (SMS) can be utilised as between nodes, or simply situations where an endpoint has no possibility to communicate over UDP. Consequently in addition to UDP and DTLS, alternative transport channels for conveying CoAP messages should be considered.

Extending CoAP over alternative transports allows CoAP implementations to have a significantly larger relevance in constrained as well as non-constrained networked environments: it leads to better code optimisation in constrained nodes and broader implementation reuse across new transport channels. As opposed to implementing new resource retrieval mechanisms, an application in an end-node can continue relying on using CoAP's REST-based resource retrieval and manipulation for this purpose, while changes in endpoint identification and the transport protocol can be addressed by a transport-specific messaging sublayer. This simplifies development and memory requirements. Resource representations are also visible in an end-to-end manner for any CoAP client. In certain conditions, the processing and computational overhead for conveying CoAP Requests and Responses from one underlying transport to another, would be less than that of an application-level gateway performing protocol translation of individual messages between CoAP and another resource retrieval protocol such as HTTP.

This document first provides scenarios where usage of CoAP over alternative transports is either currently underway, or may prove advantageous in the future. A simple transport type classification for CoAP-capable nodes is provided next. Then a new URI format is described through which a CoAP resource representation can be formulated that expresses transport identification in addition to endpoint information and resource paths. Following that, a discussion of the various transport properties which influence how CoAP Request and Response messages are mapped to transport level payloads, is presented.

This document however, does not touch on application QoS requirements, user policies or network adaptation, nor does it advocate replacing the current practice of UDP-based CoAP communication.

2. Usage Cases

Apart from UDP and DTLS, CoAP usage is being specified for the following environments as of this writing:

2.1. Use of SMS

CoAP messages can be sent via SMS between CoAP end-points in a cellular network [I-D.becker-core-coap-sms-gprs]. A CoAP Request message can also be sent via SMS from a CoAP client to a sleeping CoAP Server as a wake-up mechanism and trigger communication via IP. For this reason, the Open Mobile Alliance (OMA) specifies both UDP and SMS as transports for M2M communication in cellular networks. The OMA Lightweight M2M (LWM2M) protocol being drafted uses CoAP, and as transports, specifies both UDP as well as Short Message Service (SMS) bindings [OMALWM2M]. DTLS is being proposed for securing CoAP messages over SMS between Mobile Stations [I-D.fossati-dtls-over-gsm-sms].

2.2. Use of WebSockets

The WebSocket protocol has been proposed as a transport channel between WebSocket enabled CoAP end-points on the Internet [I-D.savolainen-core-coap-websockets]. This is particularly useful to enable CoAP communication within HTML5 apps and web browsers, especially in smart devices, that do not have any means to use low-level socket interfaces. Embedded client side scripts create new WebSocket connections to various WebSocket-enabled servers, through which CoAP messages can be exchanged. This also allows a browser containing an embedded CoAP server to open a connection to a WebSocket enabled CoAP Mirror Server [I-D.vial-core-mirror-server] to register and update its resources.

2.3. Use of P2P Overlays

[I-D.jimenez-p2psip-coap-reload] specifies how CoAP nodes can use a peer-to-peer overlay network called RELOAD, as a resource caching facility for storing wireless sensor data. When a CoAP node registers its resources with a RELOAD Proxy Node (PN), the node computes a hash value from the CoAP URI and stores it as a structure together with the PN's Node ID as well as the resources. Resource retrieval by CoAP nodes is accomplished by computing the hash key over the Request URI, opening a connection to the overlay and using its message routing system to contact the CoAP server via its PN.

2.4. Use of TCP and TLS

Using TCP [I-D.ietf-core-coap-tcp-tls], allows easier communication between CoAP clients and servers separated by firewalls and NATs. This also allows CoAP messages to be transported over push notification services from a notification server to a client app on a smartphone, that may previously have subscribed to receive change notifications of CoAP resource representations, possibly by using CoAP Observe [RFC7641]. [I-D.ietf-core-coap-tcp-tls] also discusses using TLS as a transport to securely convey CoAP messages over TCP.

3. Node Types based on Transport Availability

The term "alternative transport" in this document thus far has been used to refer to any non-UDP and non-DTLS transport that can convey CoAP messages in its payload. A node however, may in fact possess the capability to utilise CoAP over multiple transport channels at its disposal, simultaneously or otherwise, at any point in time to communicate with a CoAP end-point. Such communication can obviously take place over UDP and DTLS as well. Inevitably, if two CoAP endpoints reside in distinctly separate networks with orthogonal transports, a CoAP proxy node is needed between the two networks so that CoAP Requests and Responses can be exchanged properly.

In [RFC7228], Tables 1, 3 and 4 introduced classification schemes for devices, in terms of their resource constraints, energy limitations and communication power. For this document, in addition to these capabilities, it seems useful to additionally identify devices based on their transport capabilities.

Name	Transport Availability
T0	Single transport
T1	Multiple transports, with one or more active at any point in time
T2	Multiple active transports at all times

Table 1: Classes of Available Transports

Type T0 nodes possess the capability of exactly 1 type of transport channel for CoAP, at all times. These include both active and sleepy nodes, which may choose to perform duty cycling for power saving.

Type T1 nodes possess multiple different transports, and can retrieve or expose CoAP resources over any or all of these transports. However, not all transports are constantly active and certain transport channels and interfaces could be kept in a mostly-off state for energy-efficiency, such as when using CoAP over SMS (refer to section 2.1)

Type T2 nodes possess more than 1 transport, and multiple transports are simultaneously active at all times. CoAP proxy nodes which allow CoAP endpoints from disparate transports to communicate with each other, are a good example of this.

4. CoAP Alternative Transport URI

Based on the usage scenarios as well as the transport classes presented in the preceding sections, this section discusses the formulation of a new URI format for representing CoAP resources over alternative transports.

CoAP is logically divided into 2 sublayers, whereby the upper layer is responsible for the protocol functionality of exchanging request and response messages, while the messaging layer is bound to UDP. These 2 sublayers are tightly coupled, both being responsible for properly encoding the header and body of the CoAP message. The CoAP URI is used by both logical sublayers. For a URI that is expressed generically as

URI = scheme ":" "//" authority path-abempty ["?" query]

a simple example CoAP URI, "coap://server.example.com/sensors/temperature" is interpreted as follows:

coap	://	server.example.com	/sensors/temperature
____/		_____/\	_____/\
		\	\
protocol		endpoint	parameterised
identifier		identifier	resource
			identifier

Figure 1: The CoAP URI format

The resource path is explicitly expressed, and the endpoint identifier, which contains the host address at the network-level is also directly bound to the scheme name containing the application-level protocol identifier. The choice of a specific transport for a scheme, however, cannot be embedded with a URI, but is defined by convention or standardisation of the protocol using the scheme. As examples, [RFC5092] defines the 'imap' scheme for the IMAP protocol over TCP, while [RFC2818] requires that the 'https' protocol identifier be used to differentiate using HTTP over TLS instead of TCP.

4.1. Design Considerations

Several ways of formulating a URI which express an alternative transport binding to CoAP, can be envisioned. When such a URI is provided from an application to its CoAP implementation, the URI component containing transport-specific information can be checked to allow CoAP to use the appropriate transport for a target endpoint identifier.

The following design considerations influence the formulation of a new URI expressing CoAP resources over alternative transports:

1. The CoAP Transport URI must conform to the generic syntax for a URI described in [RFC3986]. By ensuring conformance to RFC3986, the need for custom URI parsers as well as resolution algorithms can be obviated. In particular, a URI format needs to be described in which each URI component clearly meets the syntax and percent-encoding rules described.
2. A CoAP Transport URI can be supplied as a Proxy-Uri option by a CoAP end-point to a CoAP forward proxy. This allows communication with a CoAP end-point residing in a network using a different transport. Section 6.4 of [RFC7252] provides an algorithm for parsing a received URI to obtain the request's options. Conformance to [RFC3986] is also necessary in order for the parsing algorithm to be successful.
3. Request messages sent to a CoAP endpoint using a CoAP Transport URI may be responded to with a relative URI reference, for example, of the form "../..path/to/resource". In such cases, the requesting endpoint needs to resolve the relative reference against the original CoAP Transport URI to then obtain a new target URI to which a request can be sent to, to obtain a resource representation. [RFC3986] provides an algorithm to establish how relative references can be resolved against a base URI to obtain a target URI. Given this algorithm, a URI format needs to be described in which relative reference resolution does

not result in a target URI that loses its transport-specific information

4. The host component of current CoAP URIs can either be an IPv4 address, an IPv6 address or a resolvable hostname. While the usage of DNS can sometimes be useful for distinguishing transport information (see section 4.3.1), accessing DNS over some alternative transport environments may be challenging. Therefore, a URI format needs to be described which is able to represent a resource without heavy reliance on a naming infrastructure, such as DNS.

4.2. URI format

To meet the design considerations previously discussed, the transport information is expressed as part of the URI scheme component. This is performed by minting new schemes for alternative transports using the form "coap+<transport-name>" and/or "coaps+<transport-name>", where the name of the transport is clearly and unambiguously described. Each scheme name formed in this manner is used to differentiate the use of CoAP, or CoAP using DTLS, over an alternative transport respectively. The endpoint identifier, path and query components together with each scheme name would be used to uniquely identify each resource.

Examples of such URIs are:

- o coap+tcp://[2001:db8::1]:5683/sensors/temperature for using CoAP over TCP
- o coap+tls://[2001:db8::1]:5683/sensors/temperature for using CoAP over TLS
- o coaps+sctp://[2001:db8::1]:5683/sensors/temperature for using CoAP over DTLS over SCTP
- o coap+sms://0015105550101/sensors/temperature for using CoAP over SMS with the endpoint identifier being a telephone subscriber number
- o coaps+sms://0015105550101/sensors/temperature for using CoAP over DTLS over SMS with the endpoint identifier being a telephone subscriber number
- o coap+ws://www.example.com/sensors/temperature for using CoAP over WebSockets

- o `coap+wss://www.example.com/sensors/temperature` for using CoAP over secure WebSockets (WebSockets using TLS)

A URI of this format to distinguish transport types is simple to understand and not dissimilar to the CoAP URI format. As the usage of each alternative transport results in an entirely new scheme, IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [I-D.ietf-appsawg-uri-scheme-reg], particularly where permanent URI scheme registration is concerned. CoAP resources transported over UDP or DTLS must conform to Section 6 of [RFC7252] and utilise "coap" or "coaps" for the URI scheme, instead of "coap+udp" or "coap+dtls".

It is also entirely possible for each new scheme to specify its own rules for how resource and transport endpoint information can be presented. However, the URIs and resource representations arising from their usage should meet the URI design considerations and guidelines mentioned in Section 4.1. In addition, each new transport being defined should take into consideration the various transport-level properties that can have an impact on how CoAP messages are conveyed as payload. This is elaborated on in the next section.

5. Alternative Transport Analysis and Properties

In this section the various characteristics of alternative transports for successfully supporting various kinds of functionality for CoAP are considered. CoAP factors lossiness, unreliability, small packet sizes and connection statelessness into its protocol logic. General transport differences and their impact on carrying CoAP messages here are discussed.

Property 1: 1:N communication support.

This refers to the ability of the transport protocol to support broadcast and multicast communication. For example, group communication for CoAP is based on multicasting Request messages and receiving Response messages via unicast [RFC7390]. A protocol such as TCP would be ill-suited for group communications using multicast. Anycast support, where a message is sent to a well defined destination address to which several nodes belong, on the other hand, is supported by TCP.

Property 2: Transport-level reliability.

This refers to the ability of the transport protocol to support properties such as guaranteeing reliability against packet loss, ensuring ordered packet delivery and having error control. When CoAP Request and Response messages are delivered over such transports, the

CoAP implementations elide certain fields in the packet header. As an example, if the usage of a connection-oriented transport renders it unnecessary to specify the various CoAP message types, the Type field can be elided. For some connection-oriented transports, such as WebSockets, the version of CoAP being used can be negotiated during the opening transfer. Consequently, the Version field in CoAP packets can also be elided.

Property 3: Message encoding.

While parts of the CoAP payload are human readable or are transmitted in XML, JSON or SenML format, CoAP is essentially a low overhead binary protocol. Efficient transmission of such packets would therefore be met with a transport offering binary encoding support. Techniques exist in allowing binary payloads to be transferred over text-based transport protocols such as base-64 encoding. When using SMS as a transport, for example, although binary encoding is supported, Appendix A.5 of [I-D.bormann-coap-misc] indicates binary encoding for SMS may not always be viable. A fuller discussion about performing CoAP message encoding for SMS can be found in Appendix A.5 of [I-D.bormann-coap-misc]

Property 4: Network byte order.

CoAP, as well as transports based on the IP stack use a Big Endian byte order for transmitting packets over the air or wire, while transports based on Bluetooth and Zigbee prefer Little Endian byte ordering for packet fields and transmission. Any CoAP implementation that potentially uses multiple transports has to ensure correct byte ordering for the transport used.

Property 5: MTU correlation with CoAP PDU size.

Section 4.6 of [RFC7252] discusses the avoidance of IP fragmentation by ensuring CoAP message fit into a single UDP datagram. End-points on constrained networks using 6LoWPAN may use blockwise transfers to accommodate even smaller packet sizes to avoid fragmentation. The MTU sizes for Bluetooth Low Energy as well as Classic Bluetooth are provided in Section 2.4 of [RFC7668]. Transport MTU correlation with CoAP messages helps ensure minimal to no fragmentation at the transport layer. On the other hand, allowing a CoAP message to be delivered using a delay-tolerant transport service such as the Bundle Protocol [RFC5050] would imply that the CoAP message may be fragmented (or reconstituted) along various nodes in the DTN as various sized bundles and bundle fragments.

Property 6: Framing

When using CoAP over a streaming transport protocol such as TCP, as opposed to datagram based protocols, care must be observed in preserving message boundaries. Commonly applied techniques at the transport level include the use of delimiting characters for this purpose as well as message framing and length prefixing.

Property 7: Transport latency.

A confirmable CoAP request would be retransmitted by a CoAP end-point if a response is not obtained within a certain time. A CoAP end-point registering to a Resource Directory uses a POST message that could include a lifetime value. A sleepy end-point similarly uses a lifetime value to indicate the freshness of the data to a CoAP Mirror Server. Care needs to be exercised to ensure the latency of the transport being used to carry CoAP messages is small enough not to interfere with these values for the proper operation of these functionalities.

Property 8: Connection Management.

A CoAP endpoint using a connection-oriented transport should be responsible for proper connection establishment prior to sending a CoAP Request message. Both communicating endpoints may monitor the connection health during the Data Transfer phase. Finally, once data transfer is complete, at least one end point should perform connection teardown gracefully.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

New security risks are not envisaged to arise from the guidelines given in this document, for describing a new URI format containing transport identification within the URI scheme component. However, when specific alternative transports are selected for implementing support for carrying CoAP messages, risk factors or vulnerabilities can be present. Examples include privacy trade-offs when MAC addresses or phone numbers are supplied as URI authority components, or if specific URI path components employed for security-specific interpretations are accidentally encountered as false positives. While this document does not make it mandatory to introduce a security mode with each transport, it recommends ascribing meaning to the use of "coap+" and "coaps+" prefixes in the scheme component, with the "coaps+" prefix used for DTLS-based CoAP messages over the alternative transport.

8. Acknowledgements

The draft has benefited greatly from reviews, comments and ideas from Thomas Fossati, Akbar Rahman, Klaus Hartke, Martin Thomson, Mark Nottingham, Dave Thaler, Graham Klyne, Carsten Bormann and Markus Becker.

9. References

9.1. Normative References

- [I-D.ietf-appsawg-uri-scheme-reg]
Thaler, D., Hansen, T., Hardie, T., and L. Masinter,
"Guidelines and Registration Procedures for URI Schemes",
draft-ietf-appsawg-uri-scheme-reg-06 (work in progress),
April 2015.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, DOI 10.17487/RFC3986, January 2005,
<<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for
Constrained-Node Networks", RFC 7228,
DOI 10.17487/RFC7228, May 2014,
<<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
Application Protocol (CoAP)", RFC 7252,
DOI 10.17487/RFC7252, June 2014,
<<http://www.rfc-editor.org/info/rfc7252>>.

9.2. Informative References

- [BTCorev4.1]
BLUETOOTH Special Interest Group, "BLUETOOTH Specification
Version 4.1", December 2013.
- [I-D.becker-core-coap-sms-gprs]
Becker, M., Li, K., Kuladinithi, K., and T. Poetsch,
"Transport of CoAP over SMS", draft-becker-core-coap-sms-
gprs-05 (work in progress), August 2014.
- [I-D.bormann-coap-misc]
Bormann, C. and K. Hartke, "Miscellaneous additions to
CoAP", draft-bormann-coap-misc-27 (work in progress),
November 2014.

- [I-D.fossati-dtls-over-gsm-sms]
Fossati, T. and H. Tschofenig, "Datagram Transport Layer Security (DTLS) over Global System for Mobile Communications (GSM) Short Message Service (SMS)", draft-fossati-dtls-over-gsm-sms-01 (work in progress), October 2014.
- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Technologies, Z., and H. Tschofenig, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-tcp-tls-01 (work in progress), November 2015.
- [I-D.jimenez-p2psip-coap-reload]
Jimenez, J., Lopez-Vega, J., Maenpaa, J., and G. Camarillo, "A Constrained Application Protocol (CoAP) Usage for REsource LOcation And Discovery (RELOAD)", draft-jimenez-p2psip-coap-reload-10 (work in progress), July 2015.
- [I-D.savolainen-core-coap-websockets]
Savolainen, T., Hartke, K., and B. Silverajan, "CoAP over WebSockets", draft-savolainen-core-coap-websockets-05 (work in progress), October 2015.
- [I-D.vial-core-mirror-server]
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-server-01 (work in progress), April 2013.
- [OMALWM2M]
Open Mobile Alliance (OMA), "Lightweight Machine to Machine Technical Specification Version 1.0", 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates and Service: Schemes", RFC 2609, DOI 10.17487/RFC2609, June 1999, <<http://www.rfc-editor.org/info/rfc2609>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.

- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <<http://www.rfc-editor.org/info/rfc4838>>.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, DOI 10.17487/RFC5050, November 2007, <<http://www.rfc-editor.org/info/rfc5050>>.
- [RFC5092] Melnikov, A., Ed. and C. Newman, "IMAP URL Scheme", RFC 5092, DOI 10.17487/RFC5092, November 2007, <<http://www.rfc-editor.org/info/rfc5092>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<http://www.rfc-editor.org/info/rfc6455>>.
- [RFC6568] Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, DOI 10.17487/RFC6568, April 2012, <<http://www.rfc-editor.org/info/rfc6568>>.
- [RFC6733] Fajardo, V., Ed., Arkko, J., Loughney, J., and G. Zorn, Ed., "Diameter Base Protocol", RFC 6733, DOI 10.17487/RFC6733, October 2012, <<http://www.rfc-editor.org/info/rfc6733>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<http://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.
- [RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <<http://www.rfc-editor.org/info/rfc7668>>.
- [WWWArchv1] <http://www.w3.org/TR/webarch/#uri-aliases>, "Architecture of the World Wide Web, Volume One", December 2004.

Appendix A. Expressing transport in the URI in other ways

Other means of indicating the transport as a distinguishable component within the CoAP URI are possible, but have been deemed unsuitable by not meeting the design considerations listed, or are incompatible with existing practices outlined in [RFC7252]. They are however, retained in this section for historical documentation and completeness.

A.1. Transport information as part of the URI authority

A single URI scheme, "coap-at" can be introduced, as part of an absolute URI which expresses the transport information within the authority component. One approach is to structure the component with a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint_identifier".

Examples of resulting URIs are:

- ```
o coap-at://tcp-server.example.com/sensors/temperature
o coap-at://sms-0015105550101/sensors/temperature
```

An implementation note here is that some generic URI parsers will fail when encountering a URI such as "coap-at://tcp-[2001:db8::1]/sensors/temperature". Consequently, an equivalent, but parseable URI from the ip6.arpa domain needs to be formulated instead. For [2001:db8::1] using TCP, this would result in the following URL:

```
coap-at://tcp-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0
.1.0.2.ip6.arpa:5683/sensors/temperature
```

Usage of an IPv4-mapped IPv6 address such as `::ffff.192.100.0.1` can similarly be expressed with a URI from the `ip6.arpa` domain.

This URI format allows the usage of a single scheme to represent multiple types of transport end-points. Consequently, it requires consistency in ensuring how various transport-specific endpoints are identified, as a single URI format is used. Attention must be paid towards the syntax rules and encoding for the URI host component. Additionally, against a base URI of the form "coap-at://tcp-server.example.com/sensors/temperature", resolving a relative reference, such as "//example.net/sensors/temperature" would result in the target URI "coap-at://example.net/sensors/temperature", in which transport information is lost.

#### A.1.1. Usage of DNS records

DNS names can be used instead of IPv6 address literals to mitigate lengthy URLs referring to the ip6.arpa domain, if usage of DNS is possible.

DNS SRV records can also be employed to formulate a URL such as:

```
coap-at://srv-_coap._tcp.example.com/sensors/temperature
```

in which the "srv" prefix is used to indicate that a DNS SRV lookup should be used for \_coap.\_tcp.example.com, where usage of CoAP over TCP is specified for example.com, and is eventually resolved to a numerical IPv4 or IPv6 address.

#### A.2. Making CoAP Resources Available over Multiple Transports

The CoAP URI used thus far is as follows:

```
URI = scheme ":" hier-part ["?" query]
hier-part = "//" authority path-abempty
```

A new URI format could be introduced, that does not possess an "authority" component, and instead defining "hier-part" to instead use another component, "path-rootless", as specified by RFC3986 [RFC3986]. The partial ABNF format of this URI would then be:

```
URI = scheme ":" hier-part ["?" query]
hier-part = path-rootless
path-rootless = segment-nz *("/" segment)
```

The full syntax of "path-rootless" is described in [RFC3986]. A generic URI defined this way would conform to the syntax of [RFC3986], while the path component can be treated as an opaque string to indicate transport types, endpoints as well as paths to CoAP resources. A single scheme can similarly be used.

A constrained node that is capable of communicating over several types of transports (such as UDP, TCP and SMS) would be able to convey a single CoAP resource over multiple transports. This is also beneficial for nodes performing caching and proxying from one type of transport to another.

Requesting and retrieving the same CoAP resource representation over multiple transports could be rendered possible by prefixing the transport type and endpoint identifier information to the CoAP URI. This would result in the following example representation:

```
coap-at:tcp://example.com?coap://example.com/sensors/temperature
 _____/ _____/
 \ / \ /
 Transport-specific CoAP Resource
 Prefix
```

Figure 2: Prefixing a CoAP URI with TCP transport

Such a representation would result in the URI being decomposed into its constituent components, with the CoAP resource residing within the query component as follows:

Scheme: coap-at

Path: tcp://example.com

Query: coap://example.com/sensors/temperature

The same CoAP resource, if requested over a WebSocket transport, would result the following URI:

```
coap-at:ws://example.com/endpoint?coap://example.com/sensors/temperature
 _____/ _____/
 \ / \ /
 Transport-specific CoAP Resource
 Prefix
```

Figure 3: Prefixing a CoAP URI with WebSocket transport

While the transport prefix changes, the CoAP resource representation remains the same in the query component:

Scheme: coap-at

Path: ws://example.com/endpoint

Query: coap://example.com/sensors/temperature

The URI format described here overcomes URI aliasing [WWWArchv1] when multiple transports are used, by ensuring each CoAP resource representation remains the same, but is prefixed with different transports. However, against a base URI of this format, resolving relative references of the form `"//example.net/sensors/temperature"` and `"/sensor2/temperature"` would again result in target URIs which lose transport-specific information.

Implementation note: While square brackets are disallowed within the path component, the '[' and ']' characters needed to enclose a literal IPv6 address can be percent-encoded into their respective equivalents. The ':' character does not need to be percent-encoded. This results in a significantly simpler URI string compared to section 2.2, particularly for compressed IPv6 addresses. Additionally, the URI format can be used to specify other similar address families and formats, such as Bluetooth addresses [BTCOREv4.1].

### A.3. Transport as part of a 'service:' URL scheme

The "service:" URL scheme name was introduced in [RFC2609] and forms the basis of service description used primarily by the Service Location Protocol. An abstract service type URI would have the form

`"service:<abstract-type>:<concrete-type>"`

where <abstract-type> refers to a service type name that can be associated with a variety of protocols, while the <concrete-type> then providing the specific details of the protocol used, authority and other URI components.

Adopting the "service:" URL scheme to describe CoAP usage over alternative transports would be rather trivial. To use a previous example, a CoAP service to discover a Resource Directory and its base RD resource using TCP would take the form

`service:coap:tcp://host.example.com/.well-known/core?rt=core-rd`

The syntax of the "service:" URL scheme differs from the generic URI syntax and therefore such a representation should be treated as an opaque URI as Section 2.1 of [RFC2609] recommends.



Authors' Addresses

Bilhanan Silverajan  
Tampere University of Technology  
Korkeakoulunkatu 10  
FI-33720 Tampere  
Finland

Email: [bilhanan.silverajan@tut.fi](mailto:bilhanan.silverajan@tut.fi)

Teemu Savolainen  
Nokia  
Hermiankatu 12 D  
FI-33720 Tampere  
Finland

Email: [teemu.savolainen@nokia.com](mailto:teemu.savolainen@nokia.com)

CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: September 6, 2018

B. Silverajan  
Tampere University of Technology  
T. Savolainen  
Nokia Technologies  
March 5, 2018

CoAP Communication with Alternative Transports  
draft-silverajan-core-coap-alternative-transports-11

Abstract

The aim of this document is to provide a way forward to best decide upon how alternative transport information can be expressed in a CoAP URI. This draft examines the requirements for a new URI format for representing CoAP resources over alternative transports. Various potential URI formats are presented. Benefits and drawbacks of embedding alternative transport information in various ways within the URI components are also discussed. From all listed formats, the document finds scheme-based model to be the most technically feasible.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 6, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|                                                                 |    |
|-----------------------------------------------------------------|----|
| 1. Introduction . . . . .                                       | 2  |
| 2. Conformance and Design Considerations . . . . .              | 4  |
| 3. Situating Transport Information in CoAP URIs . . . . .       | 5  |
| 3.1. Using the URI scheme component . . . . .                   | 5  |
| 3.1.1. Analysis . . . . .                                       | 6  |
| 3.2. Using the URI authority component . . . . .                | 6  |
| 3.2.1. Analysis . . . . .                                       | 7  |
| 3.3. Using the URI path component . . . . .                     | 7  |
| 3.3.1. Analysis . . . . .                                       | 7  |
| 3.4. Using the URI query component . . . . .                    | 7  |
| 3.4.1. Analysis . . . . .                                       | 8  |
| 4. Discussion . . . . .                                         | 8  |
| 5. IANA Considerations . . . . .                                | 8  |
| 6. Security Considerations . . . . .                            | 9  |
| 7. Acknowledgements . . . . .                                   | 9  |
| 8. References . . . . .                                         | 9  |
| 8.1. Normative References . . . . .                             | 9  |
| 8.2. Informative References . . . . .                           | 9  |
| Appendix A. Expressing transport in the URI in other ways . . . | 10 |
| A.1. Transport information as part of the URI authority . . .   | 10 |
| A.1.1. Usage of DNS records . . . . .                           | 11 |
| A.2. Making CoAP Resources Available over Multiple Transports   | 12 |
| A.3. Transport as part of a 'service:' URL scheme . . . . .     | 13 |
| Authors' Addresses . . . . .                                    | 14 |

## 1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a lightweight, binary application layer protocol designed for constrained environments. Owing to its operating environment, CoAP uses UDP and DTLS as its underlying transports between communicating endpoints. However, with an increase in deployment experiences as well as its popularity, compelling reasons exist for extending CoAP messaging to work over alternative transports. These allow CoAP to better address firewall and NAT traversal issues, to operate in Web browser-based and HTML5 applications as well as for energy-constrained M2M communication in cellular networks. At the time of writing, these transports are:

- o TCP, TLS and Websockets [RFC8323]

- o SMS for cellular networks[I-D.becker-core-coap-sms-gprs]
- o SLIP for serial interfaces[I-D.bormann-t2trg-slipmux]

CoAP uses a REST-based model similar to HTTP, where URIs are used to identify resources at servers. An important factor of allowing CoAP communication over alternative transports, is to express not only the resource identifier, but also the alternative transport information in the URI.

CoAP URIs contain information, such as the endpoint address as well as the location of the resource hosted at the endpoint. CoAP URIs beginning with "coap://" are using UDP, while those beginning with "coaps://" are using DTLS.

```
coap :// server.example.org /sensors/temperature
 ____/ __________/ __________/
 | \ \
 URI scheme URI authority URI path
```

Figure 1: A CoAP URI

Figure 1 shows the structure of a simple example CoAP URI, in which the various URI components can be interpreted as follows:

- o The URI scheme component (e.g. "coap") contains an application-level identifier which typically identifies the protocol being used as well as its transport and network level protocol configurations. Such configurations are defined by convention or standardisation of the protocol using the scheme.
- o The URI authority component ("server.example.com") contains the endpoint identification, which is typically a fully qualified domain name or a network-level host address.
- o The URI path component ("/sensors/temperature") contains a parameterised resource identifier providing the location and identity of the resource at the endpoint.

In addition to these URI components, Figure 2 shows how specific queries on resource representations are provided by CoAP clients to servers, by specifying one or more URI query components in the URI.

```
coap :// server.example.org /sensors/temperature ?u=cel
 _____/
 |
 URI query
```

Figure 2: A CoAP URI with query

This document focuses on how CoAP URIs can be extended to contain information about alternative transports. For deriving the new URI format, the main design considerations are presented in the next section. Following that, various potential URIs are presented. These URIs provide examples of how transport identifiers can be situated in the URI scheme, authority, path or query components. The proposed URIs are analysed to select feasible formats while disqualifying those not meeting the design criteria.

## 2. Conformance and Design Considerations

In order to understand which URI formats are best suited for expressing transport information, certain considerations firstly need to be taken into account. Doing so eliminates URI formats that do not meet or conform to the stated requirements. The main criteria are:

1. Conformance to the generic syntax for a URI described in [RFC3986]. A URI format needs to be described in which each URI component clearly meets the syntax and percent-encoding rules described.
2. Alignment with best practices for URI design, as described in [RFC7320]. This is particularly important when it pertains to establishing or standardising the structure and usage of URIs with respect to the various URI components.
3. Request messages sent to a CoAP endpoint using a CoAP Transport URI may be responded to with a relative URI reference. [RFC3986] provides an algorithm to establish how relative references can be resolved against a base URI to obtain a target URI. Given this algorithm, a URI format needs to be described in which relative reference resolution does not result in a target URI that loses its transport-specific information
4. The URI can be supplied as a Proxy-Uri option by a CoAP end-point to a CoAP forward proxy. This allows communication with a CoAP end-point residing in a network using a different transport. Section 6.4 of [RFC7252] provides an algorithm for parsing a received URI to obtain the request's options. Conformance to

[RFC3986] is also necessary in order for the parsing algorithm to be successful.

In addition to the above mentioned requirements, where possible, the following considerations need to be borne in mind:

1. The URI format is able to represent a resource and the transport information for use in constrained environments, without requiring the presence of a naming infrastructure, such as DNS or a directory/lookup service.
  2. Alternative transport information can be easily retrieved by computationally constrained nodes. In other words, the URI format does not result in unnecessarily complex code or logic in such nodes to parse and extract the transport to be used, nor the endpoint address.
  3. URIs are designed to uniquely identify resources. When a single resource is represented with multiple URIs, URI aliasing [WWWArchv1] occurs. Avoiding URI aliasing is considered good practice.
  4. CoAP URIs do not support fragment identifiers.
3. Situating Transport Information in CoAP URIs

The following subsections aim to describe potential URI formats in which the alternative transport information is placed in various URI components.

#### 3.1. Using the URI scheme component

Expressing the transport information in the URI scheme component can be achieved by using new schemes. These can conform to an agreed-upon convention such as "coap+alternative\_transport\_name" for each new alternative transport and/or "coaps+alternative\_transport\_name" for its secure counterpart.

Examples of such URIs are:

- o coap+tcp://server.example.org/sensors/temperature for using CoAP over TCP
- o coap+sms://0015105550101/sensors/temperature for using CoAP over SMS with the endpoint identifier being a telephone subscriber number

- o coaps+tcp://server.example.org/sensors/temperature for using CoAP over TLS

#### 3.1.1. Analysis

Expressing transport information in the URI scheme delivers a URI which is human-readable and computationally as easy to parse as standard CoAP URIs, to extract transport identification information. The URI syntax conforms to [RFC3986], and relative URI resolution does not result in the loss of transport identification information. However, each new alternative transport requires minting new schemes, and IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [RFC7595]. Additionally, should a CoAP server wish to expose its resources over multiple transports (such as both UDP and TCP) , URI aliasing can occur if the URI scheme components of these multiple URIs differ in describing the same resource.

#### 3.2. Using the URI authority component

Expressing the transport information within the authority component can result in two possible URI formats.

The first approach is to structure the URI authority's host sub-component with a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint\_identifier".

Examples of resulting URIs are:

- o coap://tcp-server.example.org/sensors/temperature for using CoAP over TCP
- o coap://sms-0015105550101/sensors/temperature for using CoAP over SMS

The second approach is to hint at the alternative transport information, by explicitly specifying using the URI authority's port sub-component, thereby differentiating them from standard CoAP URIs.

Examples of resulting URIs are:

- o coap://server.example.org:5684/sensors/temperature for using CoAP over TLS
- o coap://server.example.org:80/sensors/temperature for using CoAP over WebSockets

### 3.2.1. Analysis

Embedding the transport information in the host would violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. Consequently, the host in a URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

Embedding the transport information in the port, on the other hand, would not violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. It would result in a CoAP URI that is less human-readable, but URI aliasing is minimised.

On the other hand, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form `"/server2.example.org/path/to/another/resource"`, relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

### 3.3. Using the URI path component

Should the URI path component be used, then special characters or keywords need to be supplied in the path to make the transport explicit. Here, many proposals can exist. In general however, this will result in a URI format such as:

- o `coap://server.example.org/sensors/temperature;tcp` for using CoAP over TCP, by appending the transport information at the end of the URI.

#### 3.3.1. Analysis

Embedding the transport information in the URI path directly results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form `"../..../path/to/another/resource"`, relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI path component cannot be used as a basis for a new CoAP URI for alternative transports.

### 3.4. Using the URI query component

The alternative transport information, should URI query components be used, would result in a URI format such as:



- o `coap://server.example.org/sensors/temperature?alternative-transport=wss` for using CoAP over secure WebSockets.

#### 3.4.1. Analysis

Embedding the transport information in a URI query also results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form `"../..path/to/another/resource"`, relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI query component cannot be used as a basis for a new CoAP URI for alternative transports.

#### 4. Discussion

Based on the analysis of the various options for embedding alternative transport information in a CoAP URI, the most technically feasible option is to use the URI scheme component, as described in Section 3.1. To date, this has also been the WG consensus.

A discussion with IESG members during review of [RFC8323] revealed however, that using the URI scheme to express transport information is not desirable, to avoid the proliferation of new URI schemes for the same application-layer protocol. A strategy was instead proposed to preserve the existing CoAP URI and reuse it for alternative transports, by employing a combination of UDP Confirmable messages and timeouts to determine the eventual correct transport to use between a client and server [IESG-feedback]. The undertaken strategy would have obvious implications regarding interoperability, application and protocol logic, resource usage, for both new CoAP and existing CoAP implementations and deployments. Although URI aliasing can theoretically be avoided with this approach, at the time of writing, its technical feasibility over using the simpler strategy of using URI schemes, has yet to be validated. An obvious drawback is therefore that implementers and other SDOs may choose to provisionally or permanently register new URI schemes with IANA, for CoAP over alternative transports anyway, as was done by the Open Connectivity Foundation (OCF) [CoAP-TCP-TLS-registration].

#### 5. IANA Considerations

This memo includes no request to IANA.

## 6. Security Considerations

New security risks are not envisaged to arise from the guidelines given in this document, for describing a new URI format containing transport identification within the URI scheme component. However, when specific alternative transports are selected for implementing support for carrying CoAP messages, risk factors or vulnerabilities can be present. Examples include privacy trade-offs when MAC addresses or phone numbers are supplied as URI authority components, or if specific URI path components employed for security-specific interpretations are accidentally encountered as false positives. While this document does not make it mandatory to introduce a security mode with each transport, it recommends ascribing meaning to the use of "coap+" and "coaps+" prefixes in the scheme component, with the "coaps+" prefix used for secure transports for CoAP messages.

## 7. Acknowledgements

Email discussions, comments and ideas from Thomas Fossati, Akbar Rahman, Klaus Hartke, Martin Thomson, Mark Nottingham, Dave Thaler, Graham Klyne, Carsten Bormann and Markus Becker greatly helped previous versions of this draft.

## 8. References

### 8.1. Normative References

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.

### 8.2. Informative References

- [CoAP-TCP-TLS-registration]  
, <<https://www.iana.org>>.

- [I-D.becker-core-coap-sms-gprs]  
Kuladinithi, K., Becker, M., Li, K., and T. Poetsch,  
"Transport of CoAP over SMS", draft-becker-core-coap-sms-  
gprs-06 (work in progress), February 2017.
- [I-D.bormann-t2trg-slipmux]  
Bormann, C. and T. Kaupat, "Slipmux: Using an UART  
interface for diagnostics, configuration, and packet  
transfer", draft-bormann-t2trg-slipmux-02 (work in  
progress), January 2018.
- [IESG-feedback]  
, <[https://www.ietf.org/mail-archive/web/core/current/  
msg08768](https://www.ietf.org/mail-archive/web/core/current/msg08768)>.
- [RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates  
and Service: Schemes", RFC 2609, DOI 10.17487/RFC2609,  
June 1999, <<https://www.rfc-editor.org/info/rfc2609>>.
- [RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines  
and Registration Procedures for URI Schemes", BCP 35,  
RFC 7595, DOI 10.17487/RFC7595, June 2015,  
<<https://www.rfc-editor.org/info/rfc7595>>.
- [RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K.,  
Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained  
Application Protocol) over TCP, TLS, and WebSockets",  
RFC 8323, DOI 10.17487/RFC8323, February 2018,  
<<https://www.rfc-editor.org/info/rfc8323>>.
- [WWWArchv1]  
, <<http://www.w3>>.

## Appendix A. Expressing transport in the URI in other ways

Other means of indicating the transport as a distinguishable component within the CoAP URI are possible, but have been deemed unsuitable by not meeting the design considerations listed, or are incompatible with existing practices outlined in [RFC7252]. They are however, retained in this section for historical documentation and completeness.

### A.1. Transport information as part of the URI authority

A single URI scheme, "coap-at" can be introduced, as part of an absolute URI which expresses the transport information within the authority component. One approach is to structure the component with

a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint\_identifier".

Examples of resulting URIs are:

- ```
o coap-at://tcp-server.example.com/sensors/temperature
o coap-at://sms-0015105550101/sensors/temperature
```

An implementation note here is that some generic URI parsers will fail when encountering a URI such as "coap-at://tcp-[2001:db8::1]/sensors/temperature". Consequently, an equivalent, but parseable URI from the ip6.arpa domain needs to be formulated instead. For [2001:db8::1] using TCP, this would result in the following URL:

```
coap-at://tcp-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0
.1.0.2.ip6.arpa:5683/sensors/temperature
```

Usage of an IPv4-mapped IPv6 address such as `::ffff:192.100.0.1` can similarly be expressed with a URI from the `ip6.arpa` domain.

This URI format allows the usage of a single scheme to represent multiple types of transport end-points. Consequently, it requires consistency in ensuring how various transport-specific endpoints are identified, as a single URI format is used. Attention must be paid towards the syntax rules and encoding for the URI host component. Additionally, against a base URI of the form "coap-at://tcp-server.example.com/sensors/temperature", resolving a relative reference, such as "//example.net/sensors/temperature" would result in the target URI "coap-at://example.net/sensors/temperature", in which transport information is lost.

A.1.1. Usage of DNS records

DNS names can be used instead of IPv6 address literals to mitigate lengthy URLs referring to the ip6.arpa domain, if usage of DNS is possible.

DNS SRV records can also be employed to formulate a URL such as:

```
coap-at://srv_coap_tcp.example.com/sensors/temperature
```

in which the "srv" prefix is used to indicate that a DNS SRV lookup should be used for `_coap._tcp.example.com`, where usage of CoAP over TCP is specified for `example.com`, and is eventually resolved to a numerical IPv4 or IPv6 address.

A.2. Making CoAP Resources Available over Multiple Transports

The CoAP URI used thus far is as follows:

```
URI           = scheme ":" hier-part [ "?" query ]
hier-part     = "//" authority path-abempty
```

A new URI format could be introduced, that does not possess an "authority" component, and instead defining "hier-part" to instead use another component, "path-rootless", as specified by RFC3986 [RFC3986]. The partial ABNF format of this URI would then be:

```
URI           = scheme ":" hier-part [ "?" query ]
hier-part     = path-rootless
path-rootless = segment-nz *( "/" segment )
```

The full syntax of "path-rootless" is described in [RFC3986]. A generic URI defined this way would conform to the syntax of [RFC3986], while the path component can be treated as an opaque string to indicate transport types, endpoints as well as paths to CoAP resources. A single scheme can similarly be used.

A constrained node that is capable of communicating over several types of transports (such as UDP, TCP and SMS) would be able to convey a single CoAP resource over multiple transports. This is also beneficial for nodes performing caching and proxying from one type of transport to another.

Requesting and retrieving the same CoAP resource representation over multiple transports could be rendered possible by prefixing the transport type and endpoint identifier information to the CoAP URI. This would result in the following example representation:

```
coap-at:tcp://example.com?coap://example.com/sensors/temperature
      \_____/ \_____/
        \      \
      Transport-specific CoAP Resource
        Prefix
```

Figure 3: Prefixing a CoAP URI with TCP transport

Such a representation would result in the URI being decomposed into its constituent components, with the CoAP resource residing within the query component as follows:

Scheme: coap-at

Path: tcp://example.com

Query: coap://example.com/sensors/temperature

The same CoAP resource, if requested over a WebSocket transport, would result the following URI:

coap-at:ws://example.com/endpoint?coap://example.com/sensors/temperature

Transport-specific Prefix CoAP Resource

Figure 4: Prefixing a CoAP URI with WebSocket transport

While the transport prefix changes, the CoAP resource representation remains the same in the query component:

Scheme: coap-at

Path: ws://example.com/endpoint

Query: coap://example.com/sensors/temperature

The URI format described here overcomes URI aliasing [WWWArchv1] when multiple transports are used, by ensuring each CoAP resource representation remains the same, but is prefixed with different transports. However, against a base URI of this format, resolving relative references of the form `"//example.net/sensors/temperature"` and `"/sensor2/temperature"` would again result in target URIs which lose transport-specific information.

Implementation note: While square brackets are disallowed within the path component, the '[' and ']' characters needed to enclose a literal IPv6 address can be percent-encoded into their respective equivalents. The ':' character does not need to be percent-encoded. This results in a significantly simpler URI string compared to section 2.2, particularly for compressed IPv6 addresses. Additionally, the URI format can be used to specify other similar address families and formats, such as Bluetooth addresses.

A.3. Transport as part of a 'service:' URL scheme

The "service:" URL scheme name was introduced in [RFC2609] and forms the basis of service description used primarily by the Service Location Protocol. An abstract service type URI would have the form `"service:<abstract-type>:<concrete-type>"`

where `<abstract-type>` refers to a service type name that can be associated with a variety of protocols, while the `<concrete-type>`

then providing the specific details of the protocol used, authority and other URI components.

Adopting the "service:" URL scheme to describe CoAP usage over alternative transports would be rather trivial. To use a previous example, a CoAP service to discover a Resource Directory and its base RD resource using TCP would take the form

```
service:coap:tcp://host.example.com/.well-known/core?rt=core-rd
```

The syntax of the "service:" URL scheme differs from the generic URI syntax and therefore such a representation should be treated as an opaque URI as Section 2.1 of [RFC2609] recommends.

Authors' Addresses

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

Teemu Savolainen
Nokia Technologies
Hatanpaaen valtatie 30
FI-33100 Tampere
Finland

Email: teemu.savolainen@nokia.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 11, 2016

B. Silverajan
Tampere University of Technology
March 10, 2016

CoAP Protocol Negotiation
draft-silverajan-core-coap-protocol-negotiation-02

Abstract

CoAP has been standardised as an application-level REST-based protocol. This document introduces a way forward for CoAP clients and servers to exchange resource representations when multiple transports exist at an endpoint, by agreeing upon alternate locations as well as transport and protocol configurations.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 11, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Rationale	2
3. Goals	3
4. New Link Attribute and Relation types	4
5. Examples	4
6. IANA Considerations	6
7. Security Considerations	6
8. Acknowledgements	6
9. References	6
9.1. Normative References	6
9.2. Informative References	6
Appendix A. Change Log	7
A.1. From -01 to -02	7
A.2. From -00 to -01	7
Author's Address	7

1. Introduction

In the Constrained Application Protocol (CoAP) [RFC7252], resources are uniquely represented by Uniform Resource Identifiers (URIs). Using URIs, CoAP endpoints, such as clients, origin servers and proxies, are able to exchange representations using REST-based methods. A URI in CoAP serves two purposes. Firstly, it functions as a locator, by specifying the network location of the endpoint hosting the resource, and the underlying transport used by CoAP for accessing the resource representation. Secondly, it identifies the name of the specific resource found at that endpoint together with its namespace, or resource path.

This draft proposes a new link format attribute as well as a new link relation type that together enable an origin server to serve a resource from other protocol configurations or endpoints. CoAP clients then interact with an origin server's CoRE resource discovery interface to obtain a set of links describing alternate locations of resources.

2. Rationale

Ongoing activity and discussion in CoRE has revealed the need to convey CoAP messages over not just UDP and DTLS, but also over alternative transports such as SMS [I-D.becker-core-coap-sms-gprs], TCP [I-D.ietf-core-coap-tcp-tls] and WebSockets [I-D.savolainen-core-coap-websockets]. The underlying transport to be used by CoAP is identified by the scheme component of a new URI format, as described in [I-D.silverajan-core-coap-alternative-transports].

Working group discussions and feedback for the new URI format's design criteria indicated eventually that two sets of requirements for CoAP over alternative transports were deemed to be important: The first focuses on how to express location information within the new URI format in order to reach the origin server hosting the CoAP resource, the alternative transport used as well as the path and resource name that uniquely identifies the specific resource within the server. The scope of [I-D.silverajan-core-coap-alternative-transports] is focused towards this first set of requirements, as well as an analysis of transport layer properties.

The second set of requirements pertains to accessing CoAP resources when multiple transports are present at a CoAP endpoint, by separating endpoint location information from the identification of a CoAP resource. By doing so, both CoAP clients can better discern if the same CoAP resource representation can continue to be retrieved from a CoAP server over other transports. The multiple transport problem cannot be directly solved by simply introducing a new URI format. Therefore, [I-D.silverajan-core-coap-alternative-transports] provides a categorization of CoAP nodes based on their ability to use multiple transports to convey CoAP messages, whilst its main emphasis is in providing guidance for implementing support for CoAP over an alternative transport. Instead, the issue of multiple transports for a CoAP resource is addressed in this document.

3. Goals

Should an origin server wish to serve a resource over multiple transports, a single CoAP URI cannot be used to express the identity of the resource independently of alternate underlying transports or protocol configurations. Similarly, if the server wishes to serve representations of the resource from a different endpoint and path, the URI mechanism is incapable of capturing the relationship between these alternate representations or locations.

However, providing a way to express such relationships would be useful in the following cases:

1. CoAP clients interacting with Type T1 or T2 CoAP origin servers (see Section 3 of [I-D.silverajan-core-coap-alternative-transports]) either before or during an ongoing transaction to communicate using CoAP over a different protocol configuration or alternative transport.
2. Avoiding URI aliases [WWWArchv1], where a single resource is represented with multiple URIs, without describing relations among the alternate representations.

3. Allowing intermediate nodes such as CoAP-based proxies to intelligently cache and respond to CoAP clients with the same resource representation requested over alternative transports or server endpoints.
4. Ability to separate the CoAP resource paths from web-based CoAP endpoint path in a URI.

4. New Link Attribute and Relation types

A CoAP server wishing to allow interactions with resources from multiple locations or transports can do so by specifying the Transport Type "tt" link attribute, which is an opaque string. Multiple transport types can be included in the value of this parameter, each separated by a space. In such cases, transport types appear in a prioritised list, with the most preferred transport type by the CoAP server specified first and the lowest priority transport type last.

At the same time, each transport type supported by the server is also described with an "altloc" link relation type. The "altloc" relation type specifies a URI (containing the URI scheme, authority and optionally path) providing an alternate endpoint location up to but not including the resource path of a representation.

Both "tt" and "altloc" are optional CoAP features. If supported, they occur at the granularity level of an origin server, ie. they cannot be applied selectively on some resources only. Therefore "altloc" is always anchored at the root resource ("/"). Additionally, the "tt" link attribute and "altloc" relation type can be ignored by unsupported CoAP clients.

(TBD: As type T1 nodes may not have all transports active at all times, should a lifetime value be reflected in server responses?)

5. Examples

Example 1 shows a CoAP server returning all transport types and the alternate resource locations to a CoAP client performing a CoAP Request to ./well-known/core

In this case, the server supplies two different locations to interact with resources using CoAP over TCP. At the same time, the path to the WebSocket endpoint is provided in addition to the FQDN of the server, for using CoAP over WebSockets.

```
REQ: GET /.well-known/core

RES: 2.05 Content
</sensors>;ct=40;title="Sensor Index", tt="tcp ws sms",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<coap+tcp://server.example.com/>;rel="altloc",
<coap+tcp://server.example.net/>;rel="altloc",
<coap+ws://server.example.com/ws-endpoint/>;rel="altloc",
<coap+sms://001234567/>;rel="altloc"
```

Figure 1: Example of Server response

Example 2 shows a CoAP client actively soliciting a CoAP server for all supported transport types and protocol configurations.

```
REQ: GET /.well-known/core?tt=*

RES: 2.05 Content
</sensors>;tt="tcp sms ws"
<coap+tcp://server.example.com/>;rel="altloc",
<coap+tcp://server.example.net/>;rel="altloc",
<coap+ws://server.example.com/ws-endpoint/>;rel="altloc",
<coap+sms://001234567/>;rel="altloc"
```

Figure 2: CoAP client discovering transports supported by a CoAP server.

Example 3 shows a CoAP client explicitly soliciting support for a specific transport type using a query filter parameter.

```
REQ: GET /.well-known/core?tt=sms

RES: 2.05 Content
</sensors>;tt="tcp sms ws"
<coap+sms://001234567/>;rel="altloc"
```

Figure 3: CoAP client looking for a specific transport to use with a CoAP server.

6. IANA Considerations

New link attributes and link relations need to be registered.

7. Security Considerations

Probably lots. (TBD)

8. Acknowledgements

Thanks to Klaus Hartke for comments and reviewing this draft, and Teemu Savolainen for initial discussions about protocol negotiations and lifetime values.

9. References

9.1. Normative References

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

9.2. Informative References

- [I-D.becker-core-coap-sms-gprs]
Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-05 (work in progress), August 2014.
- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Technologies, Z., and H. Tschofenig, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-tcp-tls-01 (work in progress), November 2015.
- [I-D.savolainen-core-coap-websockets]
Savolainen, T., Hartke, K., and B. Silverajan, "CoAP over WebSockets", draft-savolainen-core-coap-websockets-05 (work in progress), October 2015.
- [I-D.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-09 (work in progress), December 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

[WWWArchv1] <http://www.w3.org/TR/webarch/#uri-aliases>, "Architecture of the World Wide Web, Volume One", December 2004.

Appendix A. Change Log

A.1. From -01 to -02

Document lifetime extension, references updated

A.2. From -00 to -01

Reworked "Introduction" section, added "Rationale", and "Goals" sections.

Author's Address

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

B. Silverajan
TUT
M. Ocak
Ericsson
July 2, 2018

CoAP Protocol Negotiation
draft-silverajan-core-coap-protocol-negotiation-09

Abstract

CoAP has been standardised as an application-level REST-based protocol. When multiple transport protocols exist for exchanging CoAP resource representations, this document introduces a way forward for CoAP endpoints as well as intermediaries to agree upon alternate transport and protocol configurations as well as URIs for CoAP messaging. Several mechanisms are proposed: Extending the CoRE Resource Directory with new parameter types, introducing a new CoAP Option with which clients can interact directly with servers without needing the Resource Directory, and finally a new CoRE Link Attribute allowing exposing alternate locations on a per-resource basis.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Aim	4
2.1. Overcoming Middlebox Issues	4
2.2. Better resource caching and serving in proxies	5
2.3. Interaction with Energy-constrained Servers	6
3. Node Types based on Transport Availability	7
4. New Resource Directory Parameters	8
4.1. The 'at' RD parameter	8
4.2. The 'tt' RD parameter	10
5. CoAP Alternative-Transport Option	11
6. The 'ol' CoRE Link Attribute	14
6.1. Using /.well-known/core	14
6.2. Using CoRE Resource Directory	15
7. IANA Considerations	15
8. Security Considerations	15
9. Acknowledgements	16
10. References	16
10.1. Normative References	16
10.2. Informative References	16
Appendix A. Change Log	17
A.1. From -08 to -09	17
A.2. From -07 to -08	17
A.3. From -06 to -07	17
A.4. From -05 to -06	17
A.5. From -04 to -05	17
A.6. From -03 to -04	17
A.7. From -02 to -03	17
A.8. From -01 to -02	18
A.9. From -00 to -01	18
Authors' Addresses	18

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] allows clients, origin servers and proxies, to exchange and manipulate resource representations using REST-based methods over UDP or DTLS. CoAP messaging however can use other alternative underlying transports [I-D.silverajan-core-coap-alternative-transports].

When CoAP-based endpoints and proxies possess the ability to perform CoAP messaging over multiple transports, significant benefits can be obtained if communicating client endpoints can discover that multiple transport bindings may exist on an origin server over which CoAP resources can be retrieved. This allows a client to understand and possibly substitute a different transport protocol configuration for the same CoAP resources on the origin server, based on the preferences of the communicating peers. Inevitably, if two CoAP endpoints reside in distinctly separate networks with orthogonal transports, a CoAP proxy node is needed between the two networks so that CoAP Requests and Responses can be exchanged properly.

A URI in CoAP, however, serves two purposes simultaneously. It firstly functions as a locator, by specifying the network location of the endpoint hosting the resource, and the underlying transport used by CoAP for accessing the resource representation. It secondly identifies the name of the specific resource found at that endpoint together with its namespace, or resource path. A single CoAP URI cannot be used to express the identity of the resource independently of alternate underlying transports or protocol configuration. Multiple URIs can result for a single CoAP resource representations if:

- o the authority components of the URI differ, owing to the same physical host exposing several network endpoints. For example, "coap://example.org/sensors/temperature" and "coap://example.net/sensors/temperature"
- o the scheme components of the URI differ, owing to the origin server exposing several underlying transport alternatives. For example, "coap://example.org/sensors/temperature" and "coap+tcp://example.org/sensors/temperature"

Without a priori knowledge, clients would be unable to ascertain if two or more URIs provided by an origin server are associated to the same representation or not. Consequently, a communication mechanism needs to be conceived to allow an origin server to properly capture the relationship between these alternate representations or locations and then subsequently supply this information to clients. This also goes some way in limiting URI aliasing [WWWArchv1].

In order to support CoAP clients, proxies and servers wishing to use CoAP over multiple transports, this draft proposes the following:

- o An ability for servers to register supported CoAP transports to a CoRE Resource Directory [I-D.ietf-core-resource-directory] with optional registration lifetime values

- o A means for CoAP clients to interact with a CoRE resource directory interface for requesting and discovering alternative transports and locations of CoAP resources
- o New Resource Directory parameter types enabling the above-mentioned features.
- o A new CoAP Option called Alternative-Transport that can be used by CoAP clients to discover and retrieve the types of alternative transports available at the origin server, as well as the links describing the transport-specific endpoint address at which CoAP resources are exposed from.
- o A new CoRE Link attribute for exposing transports and endpoint locations on an origin server on a per-resource basis.

2. Aim

The following simple scenarios aim to better portray how CoAP protocol negotiation benefits communicating nodes

2.1. Overcoming Middlebox Issues

Discovering which transports are available is important for a client to determine the optimal alternative to perform CoAP messaging according to its needs, particularly when separated from a CoAP server via a NAT. It is well-known that some firewalls as well as many NATs, particularly home gateways, hinder the proper operation of UDP traffic. NAT bindings for UDP-based traffic do not have as long timeouts as TCP-based traffic.

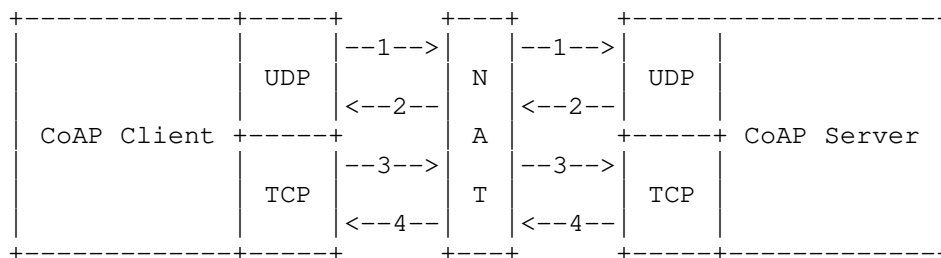


Figure 1: CoAP Client initially accesses CoAP Server over UDP and then switching to TCP

Figure 1 depicts such a scenario, where a CoAP client residing behind a NAT uses UDP initially for accessing a CoAP Server, and engages in discovering alternative transports offered by the server. The client subsequently decides to use TCP for CoAP messaging instead of UDP to set up an Observe relationship for a resource at the CoAP Server, in order to avoid incoming packets containing resource updates being discarded by the NAT.

2.2. Better resource caching and serving in proxies

Figure 2 outlines a more complex example of intermediate nodes such as CoAP-based proxies to intelligently cache and respond to CoAP or HTTP clients with the same resource representation requested over alternative transports or server endpoints. As with the earlier example, the CoAP Server registers its transports to a Resource Directory (This is assumed to be performed beforehand and not depicted in the figure, for brevity)

In this example, a CoAP over WebSockets client successfully obtains a response from a CoAP forward proxy to retrieve a resource representation from an origin server using UDP, by supplying the CoAP server's endpoint address and resource in a Proxy-URI option. Arrow 1 represents a GET request to "coap+ws://proxy.example.com" which subsequently retrieves the resource from the CoAP server using the URI "coap://example.org/sensors/temperature", shown as arrow 2.

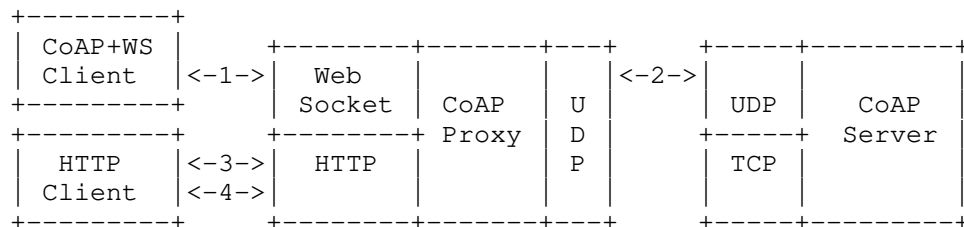


Figure 2: Proxying and returning a resource's alternate cached representations to multiple clients

Subsequently, assume an HTTP client requests the same resource, but instead specifies a CoAP over TCP alternative URI instead. Arrow 3 represents this event, where the HTTP client performs a GET request to "http://proxy.example.com/coap+tcp://example.org/sensors/temperature". When the proxy receives the request, instead of immediately retrieving the temperature resource again over TCP, it

first verifies either from the Resource Directory or directly from the server, whether the cached resource retrieved over UDP is a valid equivalent representation of the resource requested by the HTTP client over TCP. Upon confirmation, the proxy is able to supply the same cached representation to the HTTP client as well (arrow 4).

2.3. Interaction with Energy-constrained Servers

Figure 3 illustrates discovery and communication between a CoAP client and an energy-constrained CoAP Server. Such a server aims at conserving its energy unless a need arises otherwise. The figure first depicts the server registering itself to a Resource Directory over IP, and also supplies its alternative CoAP transport endpoints (in this case, SMS), in steps 1 and 2. The server can subsequently disable communication radio interfaces requiring greater energy (such as for IP-based communication), powering it up sporadically for maintenance activities like registration renewals. At other times, it maintains communication in a low-power state by listening only for incoming SMS messages.

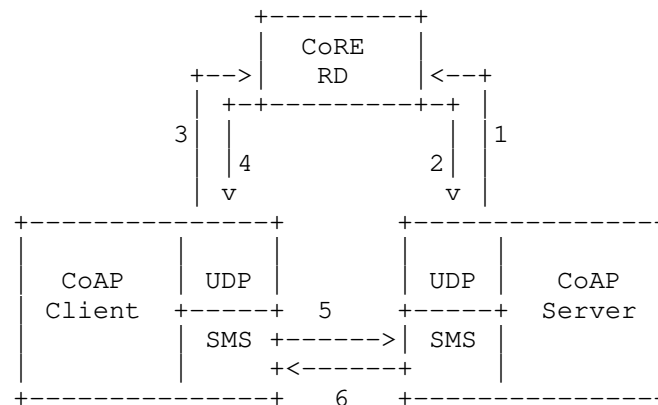


Figure 3: CoAP client interacting with RD to discover a server's SMS-based endpoint

A CoAP client wishing to perform CoAP operations with an energy-constrained CoAP server may query a resource directory for the SMS-based endpoint of the server (steps 3 and 4). Subsequently, SMS-based CoAP communication can occur between the endpoints as shown by arrows 5 and 6. Alternatively, the incoming SMS can be also used by the server as a triggering event to temporarily power up its radio

interface so that UDP or other transport-based CoAP communication can instead be employed for low latency communication with the client.

3. Node Types based on Transport Availability

In [RFC7228], Tables 1, 3 and 4 introduced classification schemes for devices, in terms of their resource constraints, energy limitations and communication power. For this document, in addition to these capabilities, it seems useful to also identify devices based on their transport capabilities.

Name	Transport Availability
T0	Single transport
T1	Multiple transports, with one or more active at any point in time
T2	Multiple active and persistent transports at all times

Table 1: Classes of Available Transports

Type T0 nodes possess the capability of exactly 1 type of transport channel for CoAP, at all times. These include both active and sleepy nodes, which may choose to perform duty cycling for power saving.

Type T1 nodes possess multiple different transports, and can retrieve or expose CoAP resources over any or all of these transports. However, not all transports are constantly active and certain transport channels and interfaces could be kept in a mostly-off state for energy-efficiency, such as when using CoAP over SMS.

Type T2 nodes possess more than 1 transport, and multiple transports are simultaneously active at all times in a persistent manner. CoAP proxy nodes which allow CoAP endpoints from disparate transports to communicate with each other, are a good example of this.

4. New Resource Directory Parameters

In order to allow resource interactions between clients and servers with multiple locations or transports, the registration, update and lookup interfaces of the CoRE Resource Directory need to be extended. In this section two new RD parameters, "at" and "tt" are introduced. Both are optional CoAP features. If supported, they occur at the granularity level of an origin server, ie. they cannot be applied selectively on some resources only. When absent, it is assumed that the server does not support multiple transports or locations.

4.1. The 'at' RD parameter

A CoAP server wishing to advertise its resources over multiple transports does so by using one or more "at" parameters to register CoAP alternative transport URIs with a Resource Directory. Such a URI would contain the scheme, address as well as any port or paths at which the server is available.

Name	Query	Validity	Description	Value
CoAP Transport URI	at	URI	URI scheme, address port and path on the server	xsd:string

Table 2: The "at" RD parameter

The "at" parameter extends the Resource Directory's Registration and Update interfaces.

The following example shows a type T1 endpoint registering its resources and advertising its ability to use TCP and WebSockets as alternative transports:

```
Req: POST coap://rd.example.com/rd?ep=node1
    &at=coap+tcp://[2001:db8:f1::2]&at=coap+ws://server.example.com
Content-Format: 40
Payload:
</temperature>;ct=0;rt="temperature";if="core.s"

Res: 2.01 Created
Location: /rd/1234
```

An endpoint lookup would just reflect the registered attributes:

Req: GET /rd-lookup/ep

Res: 2.05 Content

```
</rd/1234>;ep="node1";base="coap://[2001:db8:f1::2]:5683";  
  at="coap+tcp://[2001:db8:f1::2]";at="coap+ws://server.example.com"
```

The next example shows the same endpoint updating its registration with a new lifetime and the availability of a single alternative transport for CoAP (in this case TCP):

Req: POST /rd/1234?lt=600

&at=coap+tcp://[2001:db8:f1::2]

Content-Format: 40

Payload:

```
</temperature>;ct=0;rt="temperature";if="core.s"
```

Res: 2.04 Changed

If a lookup is performed on the same endpoint only 1 alternative transport is indicated:

Req: GET /rd-lookup/ep

Res: 2.05 Content

```
</rd/1234>;ep="node1";base="coap://[2001:db8:f1::2]:5683";  
  at="coap+tcp://[2001:db8:f1::2]"
```

A resource lookup for UDP client would be returned as the following:

Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content

```
<coap://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";if="core.s";  
  anchor="coap://[2001:db8:f1::2]"
```

A resource lookup for TCP client would be returned as the following:

Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content

```
<coap+tcp://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";if="core.s";  
  anchor="coap+tcp://[2001:db8:f1::2]"
```

4.2. The 'tt' RD parameter

A CoAP client wishing to perform a look-up on the Resource Directory for CoAP servers supporting multiple transports does so by using one or more "tt" parameters to query for CoAP alternative transport URIs.

Name	Query	Validity	Description	Value
CoAP Transport Type	tt		Transport type requested by the client	xsd:string

Table 3: The "tt" RD parameter

The "tt" parameter extends the Resource Directory's rd-lookup interface. The "tt" parameter queries existing registrations, and MUST NOT be used with the Resource Directory's registration and update interfaces.

The following example shows a client performing a lookup for endpoints supporting TCP:

Req: GET /rd-lookup/ep?tt="coap+tcp"

Res: 2.05 Content

</rd/1234>;at="coap+tcp://[2001:db8:f1::2]";ep="node1";ct="40"

The following example shows a client performing a resource lookup for endpoints supporting TCP:

Req: GET /rd-lookup/res?rt=temperature&tt="coap+tcp"

Res: 2.05 Content

<coap+tcp://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";
if="core.s";anchor="coap+tcp://[2001:db8:f1::2]"

The following example shows a client performing a lookup for endpoints supporting SMS i.e. discovering SMS transports for sleepy nodes and using SMS to communicate with the endpoint:

Req: GET /rd-lookup/ep?et=oic.d.switch&tt="coap+sms"

Res: 2.05 Content

```
</rd/2345>;at="coap+sms://0015105550101/";ep="node5";
  et="oic.d.switch";ct="40",
</rd/4521>;at="coap+sms://0015105550202/";ep="node8";
  et="oic.d.switch";ct="40"
```

5. CoAP Alternative-Transport Option

The CoAP Alternative-Transport Option can be used by CoAP clients and CoAP servers in both Request and Response messages in constrained environments where a CoRE Resource Directory is not present.

Figure 4 depicts the properties of the Alternative-Transport Option.

No.	C	U	N	R	Name	Format	Length	Default
66		x	-	x	Alternative-Transport	string	0-1034	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Figure 4: The Alternative-Transport Option

When included in a Request message, this option is used by the client in 2 possible ways. In the first case, a CoAP client can include the Option with Length 0 to retrieve all alternative transports from a CoAP server. In response to the client, the server includes base URI for each transport in its own Option. In the second case, a CoAP client can include the Option with a specific value in a CoAP Request, and the CoAP server returns the base URI(s) for the specified transport. If the specified transport by a CoAP client returns multiple results on a CoAP server, the server returns all base URIs of the transport in the response, each base URI in its own Option.

A CoAP client can also use this Option to retrieve several transports at once by including multiple Options in the request to a CoAP server. If any of the specified transports is supported by the

server, the server returns all base URIs in its own option. There can be more than 1 result for any of the transports so that each transport base URI is still included in the response in its own option.

Figure 5 describes a simple interaction between a client and a server, in which the client uses an Alternative-Transports Option with a null value to discover and retrieve all the available transports from the server, as part of a GET operation to retrieve a resource representation. The server responds with a CoAP Response message which contains the resource representation as a payload. In addition, the server also supplies multiple Alternative-Transport Options in the message, with each Option containing the base URI for an available transport. In this case the base URIs returned for TCP-based and WebSocket transports indicate their availability over a non-standard port.

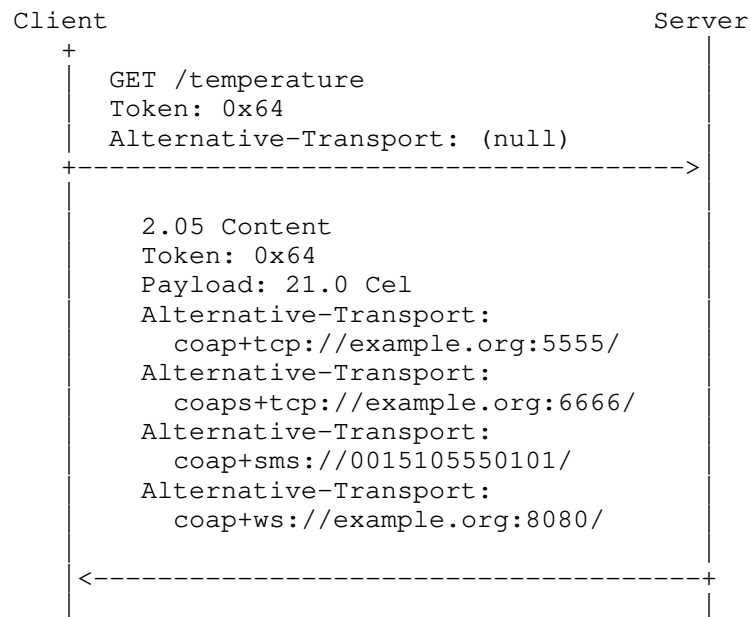


Figure 5: Requesting all available alternative transports on the server, and their locations

Alternatively, a client can also request for the availability of a specific transport on the server, as shown in Figure 6. Here, the CoAP Request contains Alternative-Transport Options with values set to request the Base URIs for TCP-based endpoints.

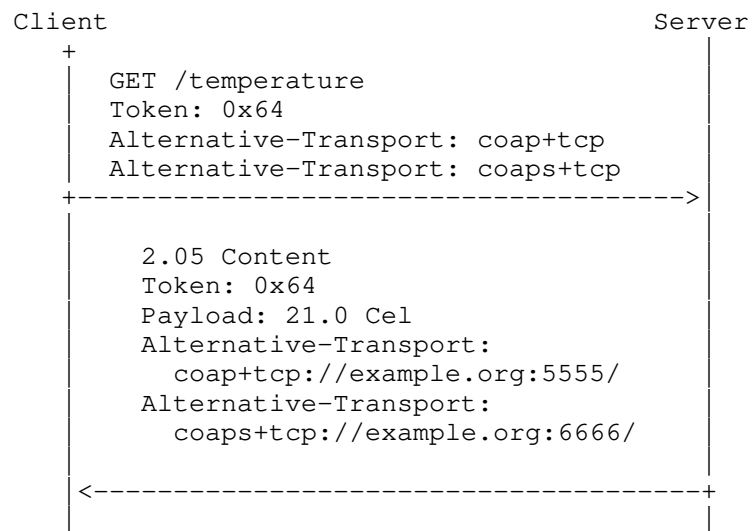


Figure 6: Requesting TCP-based alternative transports on the server, and their locations

A client may also request a subset of available transports on the server, by providing multiple Options, each having a single transport identifier. The server likewise responds to the client request by supplying the requested transport information. This is shown in Figure 7.

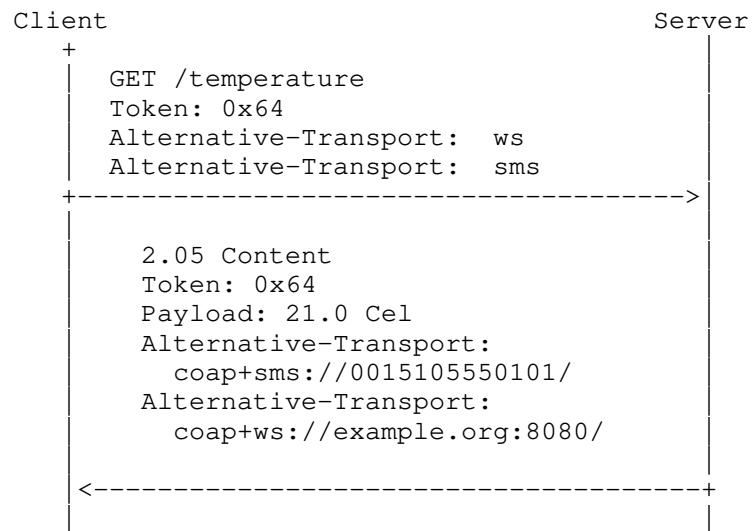


Figure 7: Requesting WebSocket- and SMS-based alternative transports on the server, and their locations

6. The 'ol' CoRE Link Attribute

In the majority of cases, it is expected that an origin server would expose all its resources uniformly on its available transports or endpoint addresses. Exceptions can exist however, where alternate locations are made available on a per-resource basis. For such cases, a new 'ol' ("other locations") attribute is provided. One or more 'ol' attributes are used to provide base URIs from which a specific resource can be reached. Allowing per-resource endpoint or transport availability enables specific functions such as firmware updates or hardware-specific operations. It also facilitates mapping to and from OCF-based resource-specific endpoint descriptions. Note that the use of 'ol' is orthogonal to using 'at' as shown in Section 6.2.

6.1. Using /.well-known/core

```
REQ: GET /.well-known/core
```

```
RES: 2.05 Content
```

```
</sensors/temp>;ct=41;rt="temperature-f";if="sensor",  
</sensors/door>;ct=41;rt="door";if="sensor",  
</sensors/light>;if="sensor"; ol="http://[FDFD::123]:61616";  
  ol="coap://server2.example.com"
```

6.2. Using CoRE Resource Directory

```
Req: POST coap:/rd.example.com/rd  
  ?ep=node1&at=coap+tcp://server.example.com&at=coap+ws://server.example.com:5  
683/ws/
```

```
Content-Format: 40
```

```
Payload:
```

```
</sensors/temp>;ct=41;rt="temperature-f";if="sensor",  
</sensors/door>;ct=41;rt="door";if="sensor",  
</sensors/light>;if="sensor"; ol="http://[FDFD::123]:61616";  
  ol="coap://server2.example.com"
```

```
Res: 2.01 Created
```

```
Location: /rd/4521
```

7. IANA Considerations

This document requests the registration of new RD parameter types "at" and "tt".

The following entry needs to be added to the CoAP Option Numbers Registry:

Number	Name	Reference
66	Alternative-Transports	(this document)

8. Security Considerations

When multiple transports, locations and representations are used, some obvious risks are present both at the origin server as well as by requesting clients.

When a client is presented with alternate URIs for retrieving resources, it presents an opportunity for attackers to mount a series of attacks, either by hijacking communication and masquerading as an alternate location or by using a man-in-the-middle attack on TLS-based communication to a server and redirecting traffic to an alternate location. A malicious or compromised server could also be used for reflective denial-of-service attacks on innocent third parties. Moreover, clients may obtain web links to alternate URIs containing weaker security properties than the existing session.

9. Acknowledgements

Thanks to Christian Amsuess, Klaus Hartke, Jaime Jimenez and Jim Schaad for comments and reviewing this draft. Teemu Savolainen was involved in initial discussions about protocol negotiations and lifetime values. Zach Shelby provided significant suggestions on how the Resource Directory can be employed and extended in place of link attributes and relation types.

10. References

10.1. Normative References

- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-14 (work in progress), July 2018.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

10.2. Informative References

- [I-D.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-11 (work in progress), March 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.

[WWWArchv1] <http://www.w3.org/TR/webarch/#uri-aliases>, "Architecture of the World Wide Web, Volume One", December 2004.

Appendix A. Change Log

A.1. From -08 to -09

Using "tt" and "Alternative Transports" updated.

A.2. From -07 to -08

Added example of energy constrained CoAP server

Updated examples of using "at" and "tt"

"at" and "ol" are no longer comma-separated URI lists.

A.3. From -06 to -07

Added support for 'ol' Link attribute

A.4. From -05 to -06

Added support for CoAP Alternative-Transports Option

A.5. From -04 to -05

Freshness update

A.6. From -03 to -04

Removed previously introduced link attribute and relation types

Initial foray with Resource Directory support

A.7. From -02 to -03

Added new author

Rewrite of "Introduction" section

Added new Aims Section

Added new Section on Node Types

Introduced "al" Active Lifetime link attribute

Added new Section on Observing transports and resources

Security and IANA considerations sections populated

A.8. From -01 to -02

Freshness update.

A.9. From -00 to -01

Reworked "Introduction" section, added "Rationale", and "Goals" sections.

Authors' Addresses

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

Mert Ocak
Ericsson
Hirsalantie 11
02420 Jorvas
Finland

Email: mert.ocak@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2016

A. Somaraju, Ed.
Tridonic GmbH & Co KG
M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov
Acklio
R. Turner
Landis+Gyr
A. Minaburo
Acklio
March 11, 2016

Structure Identifier (SID)
draft-somaraju-core-sid-00

Abstract

Structured IDentifiers (SID) are used to identify different YANG items when encoded in CBOR. This document defines the registration and assignment processes of SIDs. To enable the implementation of these processes, this document also defines a file format used to persist and publish assigned SIDs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology and Notation	2
3. Structured IDentifiers (SID)	3
4. ".sid" file lifecycle	4
5. ".sid" file format	6
6. Security Considerations	9
7. IANA Considerations	10
7.1. "SID" range registry	10
7.2. YANG module registry	11
8. Acknowledgments	12
9. References	12
9.1. Normative References	12
9.2. Informative References	12
Appendix A. ".sid" file example	13
Authors' Addresses	22

1. Introduction

This document describes the registries required to manage SIDs and a file format used to persist and publish the assigned SIDs.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in [I-D.ietf-netmod-rfc6020bis]:

- o action
- o module
- o notification
- o RPC
- o schema node

- o schema tree
- o submodule

This specification also makes use of the following terminology:

- o identifier: An identifier embodies the information required to distinguish what is being identified from all other things within its scope of identification.
- o delta : Difference between the SID assigned to the current schema node and the SID assigned to the parent.
- o item: A schema node or identity which has been allocated a SID.
- o path: A path is a string that identifies a schema node within the schema tree. A path consists of the list of schema node identifier(s) separated by slashes ("/"). Schema node identifier(s) are always listed from the top-level schema node up to the targeted schema node. (e.g. "/system-state/clock/current-datetime")

3. Structured IDentifiers (SID)

Some of the items defined in YANG [I-D.ietf-netmod-rfc6020bis] require the use of a unique identifier. In both NETCONF and RESTCONF, these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required.

This compact identifier, called SID, is encoded using an unsigned integer. To minimize its size, SIDs are often implemented using a delta from a reference SID and the current SID. To guaranty the uniqueness of each assigned SID, SID ranges MUST be registered. Section 7.1 provide more details about the registration process of SID range(s).

To avoid duplicate assignment of SIDs, the registration of the SIDs assigned to YANG module(s) is recommended. Section 7.2 provide more details about the registration process of YANG modules.

The following items are identified using SIDs:

- o identities
- o data nodes

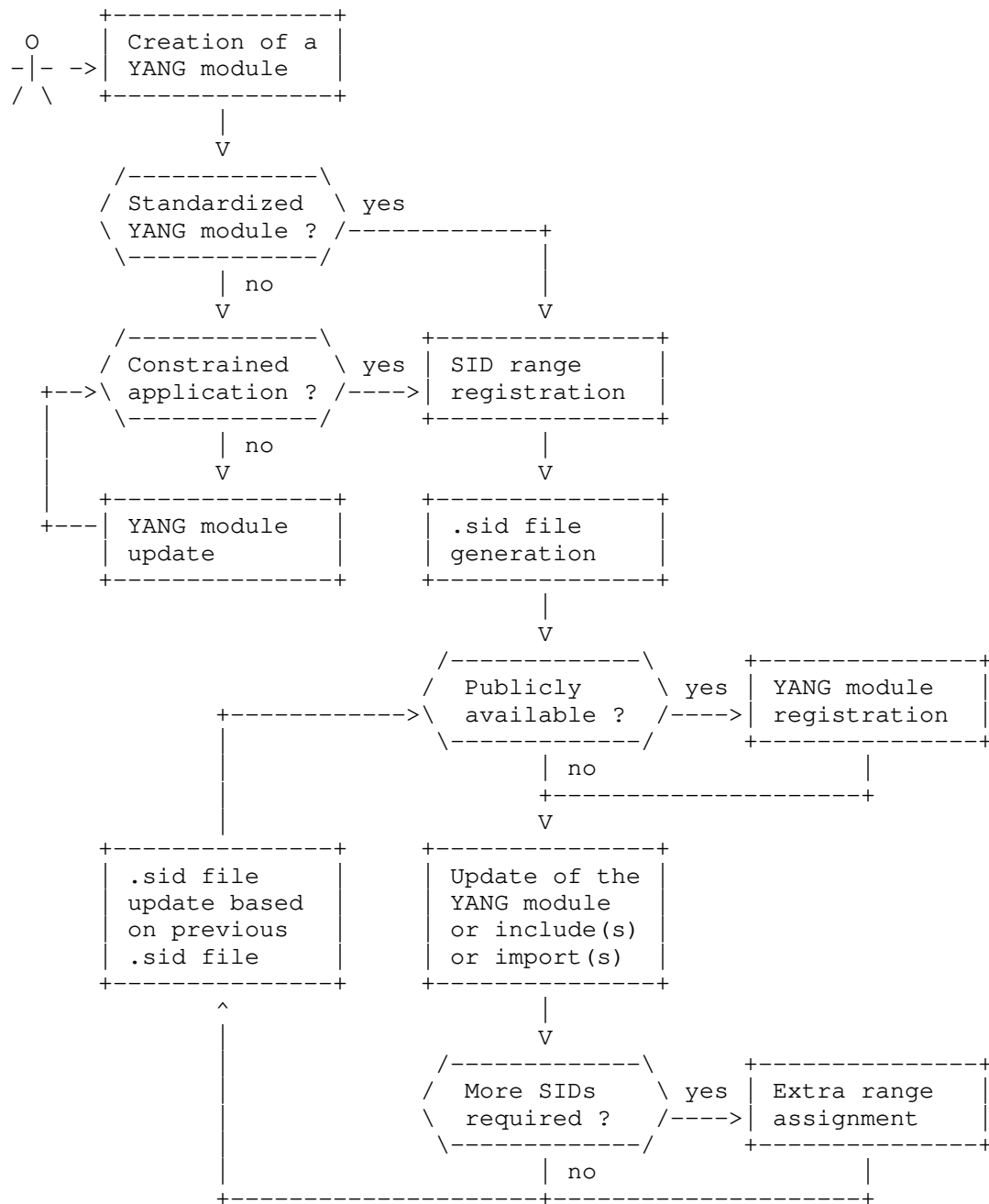
- o RPCs and associated input(s) and output(s)
- o actions and associated input(s) and output(s)
- o notifications and associated information

Assignment of SIDs can be automated, the recommended process to assign SIDs is as follows:

- o A tool extracts the different items defined for a specific YANG module.
- o The list of items is ordered by type and label.
- o SIDs are assigned sequentially for the entry point up to the size of the registered SID range. It is important to note that sequentially assigning SIDs optimizes the CBOR serialization due to the use of delta encoding.
- o If the number of items exceeds the SID range(s) allocated to a YANG module, an extra range is added for subsequent assignments.
- o SIDs are assigned permanently, items introduced by a new revision of a YANG module are added to the list of SIDs already assigned. Section 5 defines a standard file format used to store and publish SIDs.

4. ".sid" file lifecycle

The following activity diagram summarize the life cycle of ".sid" files.



YANG modules are not necessarily created in the context of constrained applications. YANG modules can be implemented using NETCONF or RESTCONF without the need to assign SIDs.

As needed, authors of YANG modules can assign SIDs to their modules. This process starts by the registration of a SID range. Once a SID range is registered, the owner of this range assigns sub-ranges to each YANG module in order to generate the associated ".sid" files. Generation of ".sid" files SHOULD be performed using an automated tool.

Registration of the .sid file associated to a YANG module is optional but recommended to promote interoperability between devices and to avoid duplicate allocation of SIDs to a single YANG module.

Each time a YANG module or one of its imported module(s) or included sub-module(s) is updated, the ".sid" file MAY need to be updated. This update SHOULD also be performed using an automated tool.

If a new revision requires more SIDs than initially allocated, a new SID range MUST be added to the assignment ranges as defined in the ".sid" file header. These extra SIDs are used for subsequent assignments.

5. ".sid" file format

".sid" files are used to persist and publish SIDs assigned to the different YANG items of a specific YANG module. The following YANG module defined the structure of this file, encoding is performed using the rules defined in [I-D.ietf-netmod-yang-json].

```
module sid-file {
  namespace "urn:ietf:ns:cool:sid-file";
  prefix sid;

  organization
    "IETF Core Working Group";

  contact
    "Ana Minaburo
    <ana@ackl.io>

    Alexander Pelov
    <mailto:a@ackl.io>

    Abhinav Somaraju
    <mailto:abhinav.somaraju@tridonic.com>
```

Laurent Toutain
<Laurent.Toutain@telecom-bretagne.eu>

Randy Turner
<mailto:Randy.Turner@landisgyr.com>

Michel Veillette
<mailto:michel.veillette@trilliantinc.com>;

```
description
  "This module define the structure of the .sid files.
  .sid files contains the identifiers (SIDs) assigned
  to the different items defined in a YANG module.
  SIDs are used to encode a data model defined in YANG
  using CBOR.";

revision 2015-12-16 {
  description
    "Initial revision.";
  reference
    "draft-veillette-core-yang-cbor-mapping";
}

typedef yang-identifier {
  type string {
    length "1..max";
    pattern '[a-zA-Z_][a-zA-Z0-9\-\_\.]*';
    pattern '\.|\.\.|\^[xX]\.*|\.[^mM]\.*|\.\.[^lL]\.*';
  }
  description
    "A YANG identifier string as defined by the 'identifier'
    rule in Section 12 of RFC 6020.";
}

typedef revision-identifier {
  type string {
    pattern '\d{4}-\d{2}-\d{2}';
  }
  description
    "Represents a date in YYYY-MM-DD format.";
}

typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?' +
      '(Z|[\+|-]\d{2}:\d{2})';
  }
}
```

```
description
  "The date-and-time type is a profile of the ISO 8601
  standard for representation of dates and times using the
  Gregorian calendar. The profile is defined by the
  date-time production in section 5.6 of RFC 3339.";
}

leaf module-name {
  type yang-identifier;
  description
    "Name of the module associated with this .sid file.";
}

leaf module-revision {
  type revision-identifier;
  description
    "Revision of the module associated with this .sid file.
    This leaf is not present if no revision statement is
    defined in the YANG module.";
}

list assignment-ranges {
  key "entry-point";
  description
    "Range(s) of SIDs available for assignment to the
    different items defined by the associated module.";

  leaf entry-point {
    mandatory true;
    type uint32;
    description
      "Lowest SID available for assignment.";
  }

  leaf size {
    mandatory true;
    type uint16;
    description
      "Number of SIDs available for assignment.";
  }
}

list items {
  key "type assigned label";
  description
    "List of items defined by the associated YANG module.";

  leaf type {
```



```
    description
      "Item type assigned, this field can be set to:
      - 'identity'
      - 'node'
      - 'notification'
      - 'rpc'
      - 'action'";
    mandatory true;
    type string {
      pattern 'identity$|node$|notification$|rpc$|action$';
    }
  }

  leaf assigned {
    mandatory true;
    type date-and-time;
    description
      "Date and time when this entry has been created.";
  }

  leaf label {
    mandatory true;
    type string;
    description
      "Label associated to this item, can be set to:
      - an identity encoded as: '<module name>:<entity name>'
      - a schema node path";
  }

  leaf sid {
    mandatory true;
    type uint32;
    description "Identifier assigned to this YANG item.";
  }
}
}
```

6. Security Considerations

The security considerations of [RFC7049] and [I-D.ietf-netmod-rfc6020bis] apply.

This document defines an new type of identifier used to encode data models defined in YANG [I-D.ietf-netmod-rfc6020bis]. As such, this identifier does not contribute to any new security issues in addition of those identified for the specific protocols or contexts for which it is used.

7. IANA Considerations

7.1. "SID" range registry

IANA is requested to create a registry for Structure Identifier (SID) ranges. This registry needs to guarantee that the ranges registered do not overlap. The registry SHALL record for each entry:

- o The entry point (first entry) of the registered SID range.
- o The size of the registered SID range.
- o The contact information of the owner of the range such as name, email address, and phone number.

The IANA policy for this registry is split into four tiers as follows:

- o The range of 0 to 9999 and 0x40000000 to 0xFFFFFFFF are reserved for future extensions of this protocol. Allocation within these ranges require IETF review or IESG approval.
- o The range of 1000 to 59999 is reserved for standardized YANG modules. Allocation within this range requires publishing of the associated ".yang" and ".sid" files. (Specification required.)
- o The range of 60000 to 99999 is reserved for experimental YANG modules. Use of this range MUST NOT be used in operational deployments since these SIDs are not globally unique which limit their interoperability.
- o The range of 100000 to 0x3FFFFFFF is available on a first come first served basis. The only information required from the registrant is a valid contact information. The recommended size of the SID ranges allocated is 1,000 for private use and 10,000 for standard development organizations (SDOs). Registrants MAY request fewer or more SIDs based on their expected, sat needs. Allocation of a significantly larger SID range MAY required IETF review or IESG approval. IANA MAY delegate this registration process to one or multiple sub-registries. The recommended size of the SID range allocation for a sub-registry is 1,000,000.

Entry Point	Size	Registration Procedures
0	1,000	IETF review or IESG approval
1,000	59,000	Specification and associated ".yang" and ".sid" files required
60,000	40,000	Experimental use
100,000	0x3ffe7960	Contact information is required. Registration of the module name(s) and associated ".yang" and ".sid" files are optional.
0x40000000	2^64-0x40000000	Specification required, expert review

7.2. YANG module registry

Each registered SID range can be used to assign SIDs to one or more YANG modules. To track which YANG modules have been assigned and to avoid duplicate allocation, IANA is requested to provide a method to register and query the following information:

- o The YANG module name
- o The contact information of the author
- o The specification reference
- o The associated ".yang" file(s) (Optional)
- o The associated ".sid" file (Optional)

Registration of YANG modules is optional. When a YANG module is registered, the registrant MUST provide the module name and contact information and/or a specification reference.

The registration of the associated ".yang" and ".sid" files is optional. When provided, the validity of the files MUST be verified. This can be accomplished by a YANG validation tool specially modified to support ".sid" file verification. The SID range specified within the ".sid" file SHOULD also be checked against the "SID" range registry (Section 7.1) and against the other YANG modules registered to detect any duplicate use of SIDs.

Initial entries in this registry are as follows:

Entry Point	Size	Module name	Reference
1000	100	ietf-cool	[I-D.veillette-core-cool]
1100	400	iana-if-type	[RFC7224]
1500	100	ietf-interfaces	[RFC7223]
1600	100	ietf-ip	[RFC7277]
1700	100	ietf-system	[RFC7317]

8. Acknowledgments

The authors would like to thank Carsten Bormann for his help during the development of this document and his useful comments during the review process.

9. References

9.1. Normative References

- [I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language",
draft-ietf-netmod-rfc6020bis-11 (work in progress),
February 2016.
- [I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG",
draft-ietf-netmod-yang-json-09 (work in progress), March
2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.

9.2. Informative References

- [I-D.veillette-core-cool]
Veillette, M. and A. Pelov, "Constrained Objects
Language", draft-veillette-core-cool-00 (work in
progress), November 2015.

- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<http://www.rfc-editor.org/info/rfc7224>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<http://www.rfc-editor.org/info/rfc7277>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.

Appendix A. ".sid" file example

The following .sid file (ietf-system@2014-08-06.sid) have been generated using the following yang modules:

- o ietf-system@2014-08-06.yang
- o ietf-yang-types@2013-07-15.yang
- o ietf-inet-types@2013-07-15.yang
- o ietf-netconf-acm@2012-02-22.yang
- o iana-crypt-hash@2014-04-04.yang

```
{
  "assignment-ranges": [
    {
      "entry-point": 1700,
      "size": 100
    }
  ],
  "module-name": "ietf-system",
  "module-revision": "2014-08-06",
  "items": [
    {
      "type": "identity",
      "assigned": "2016-01-13T21:00:19Z",
      "label": "ietf-system:authentication-method",
      "sid": 1700
    },
    {
```

```
    "type": "identity",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "ietf-system:local-users",
    "sid": 1701
  },
  {
    "type": "identity",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "ietf-system:radius",
    "sid": 1702
  },
  {
    "type": "identity",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "ietf-system:radius-authentication-type",
    "sid": 1703
  },
  {
    "type": "identity",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "ietf-system:radius-chap",
    "sid": 1704
  },
  {
    "type": "identity",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "ietf-system:radius-pap",
    "sid": 1705
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system",
    "sid": 1706
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-state",
    "sid": 1707
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-state/clock",
    "sid": 1708
  },
  {
```

```
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-state/clock/boot-datetime",
    "sid": 1709
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-state/clock/current-datetime",
    "sid": 1710
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-state/platform",
    "sid": 1711
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-state/platform/machine",
    "sid": 1712
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-state/platform/os-name",
    "sid": 1713
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-state/platform/os-release",
    "sid": 1714
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-state/platform/os-version",
    "sid": 1715
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/authentication",
    "sid": 1716
  },
  {
```

```
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/authentication/user",
    "sid": 1717
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/authentication/user-authentication-order",
    "sid": 1718
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/authentication/user/authorized-key",
    "sid": 1719
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/authentication/user/authorized-key/algorithm",
    "sid": 1720
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/authentication/user/authorized-key/key-data",
    "sid": 1721
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/authentication/user/authorized-key/name",
    "sid": 1722
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/authentication/user/name",
    "sid": 1723
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/authentication/user/password",
    "sid": 1724
  },
  {
```



```
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/clock",
    "sid": 1725
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/clock/timezone/timezone-name/timezone-name",
    "sid": 1726
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/clock/timezone/timezone-utc-offset/
    timezone-utc-offset",
    "sid": 1727
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/contact",
    "sid": 1728
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/dns-resolver",
    "sid": 1729
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/dns-resolver/options",
    "sid": 1730
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/dns-resolver/options/attempts",
    "sid": 1731
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/dns-resolver/options/timeout",
    "sid": 1732
  },
  },
```

```
{
  "type": "node",
  "assigned": "2016-01-13T21:00:19Z",
  "label": "/system/dns-resolver/search",
  "sid": 1733
},
{
  "type": "node",
  "assigned": "2016-01-13T21:00:19Z",
  "label": "/system/dns-resolver/server",
  "sid": 1734
},
{
  "type": "node",
  "assigned": "2016-01-13T21:00:19Z",
  "label": "/system/dns-resolver/server/name",
  "sid": 1735
},
{
  "type": "node",
  "assigned": "2016-01-13T21:00:19Z",
  "label": "/system/dns-resolver/server/transport/udp-and-tcp/
udp-and-tcp",
  "sid": 1736
},
{
  "type": "node",
  "assigned": "2016-01-13T21:00:19Z",
  "label": "/system/dns-resolver/server/transport/udp-and-tcp/
udp-and-tcp/address",
  "sid": 1737
},
{
  "type": "node",
  "assigned": "2016-01-13T21:00:19Z",
  "label": "/system/dns-resolver/server/transport/udp-and-tcp/
udp-and-tcp/port",
  "sid": 1738
},
{
  "type": "node",
  "assigned": "2016-01-13T21:00:19Z",
  "label": "/system/hostname",
  "sid": 1739
},
{
  "type": "node",
  "assigned": "2016-01-13T21:00:19Z",
```

```
    "label": "/system/location",
    "sid": 1740
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/ntp",
    "sid": 1741
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/ntp/enabled",
    "sid": 1742
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/ntp/server",
    "sid": 1743
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/ntp/server/association-type",
    "sid": 1744
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/ntp/server/iburst",
    "sid": 1745
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/ntp/server/name",
    "sid": 1746
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/ntp/server/prefer",
    "sid": 1747
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
```

```
    "label": "/system/ntp/server/transport/udp/udp",
    "sid": 1748
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/ntp/server/transport/udp/udp/address",
    "sid": 1749
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/ntp/server/transport/udp/udp/port",
    "sid": 1750
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius",
    "sid": 1751
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius/options",
    "sid": 1752
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius/options/attempts",
    "sid": 1753
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius/options/timeout",
    "sid": 1754
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius/server",
    "sid": 1755
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
```

```
    "label": "/system/radius/server/authentication-type",
    "sid": 1756
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius/server/name",
    "sid": 1757
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius/server/transport/udp/udp",
    "sid": 1758
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius/server/transport/udp/udp/address",
    "sid": 1759
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius/server/transport/udp/udp/
authentication-port",
    "sid": 1760
  },
  {
    "type": "node",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system/radius/server/transport/udp/udp/shared-secret",
    "sid": 1761
  },
  {
    "type": "rpc",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/set-current-datetime",
    "sid": 1762
  },
  {
    "type": "rpc",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/set-current-datetime/input/current-datetime",
    "sid": 1763
  },
  {
    "type": "rpc",
```

```
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-restart",
    "sid": 1764
  },
  {
    "type": "rpc",
    "assigned": "2016-01-13T21:00:19Z",
    "label": "/system-shutdown",
    "sid": 1765
  }
]
}
```

Authors' Addresses

Abhinav Somaraju (editor)
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: <http://www.landisgyr.com/>

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

A. Somaraju, Ed.
Tridonic GmbH & Co KG
M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov
Acklio
R. Turner
Landis+Gyr
A. Minaburo
Acklio
July 08, 2016

Structure Identifier (SID)
draft-somaraju-core-sid-01

Abstract

Structured IDentifiers (SID) are used to identify different YANG items when encoded in CBOR. This document defines the registration and assignment processes of SIDs. To enable the implementation of these processes, this document also defines a file format used to persist and publish assigned SIDs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology and Notation	2
3. Structured IDentifiers (SID)	3
4. ".sid" file lifecycle	4
5. ".sid" file format	6
6. Security Considerations	9
7. IANA Considerations	9
7.1. "SID" range registry	9
7.2. YANG module registry	11
8. Acknowledgments	11
9. References	12
9.1. Normative References	12
9.2. Informative References	12
Appendix A. ".sid" file example	13
Authors' Addresses	21

1. Introduction

This document describes the registries required to manage SIDs and a file format used to persist and publish the assigned SIDs.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in [I-D.ietf-netmod-rfc6020bis]:

- o action
- o feature
- o module
- o notification
- o RPC

- o schema node
- o schema tree
- o submodule

This specification also makes use of the following terminology:

- o identifier: An identifier embodies the information required to distinguish what is being identified from all other things within its scope of identification.
- o delta : Difference between the current SID and a reference SID. A reference SID is defined for each context for which deltas are used.
- o item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- o path: A path is a string that identifies a schema node within the schema tree. A path consists of the list of schema node identifier(s) separated by slashes ("/"). Schema node identifier(s) are always listed from the top-level schema node up to the targeted schema node. (e.g. "/system-state/clock/current-datetime")

3. Structured Identifiers (SID)

Some of the items defined in YANG [I-D.ietf-netmod-rfc6020bis] require the use of a unique identifier. In both NETCONF and RESTCONF, these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required.

This compact identifier, called SID, is encoded using an unsigned integer. To minimize its size, SIDs are often implemented using a delta from a reference SID and the current SID. To guaranty the uniqueness of each assigned SID, SID ranges MUST be registered. Section 7.1 provide more details about the registration process of SID range(s).

To avoid duplicate assignment of SIDs, the registration of the SIDs assigned to YANG module(s) is recommended. Section 7.2 provide more details about the registration process of YANG modules.

The following items are identified using SIDs:

- o identities
- o data nodes
- o RPCs and associated input(s) and output(s)
- o actions and associated input(s) and output(s)
- o notifications and associated information
- o YANG modules, submodules and features

Assignment of SIDs can be automated, the recommended process to assign SIDs is as follows:

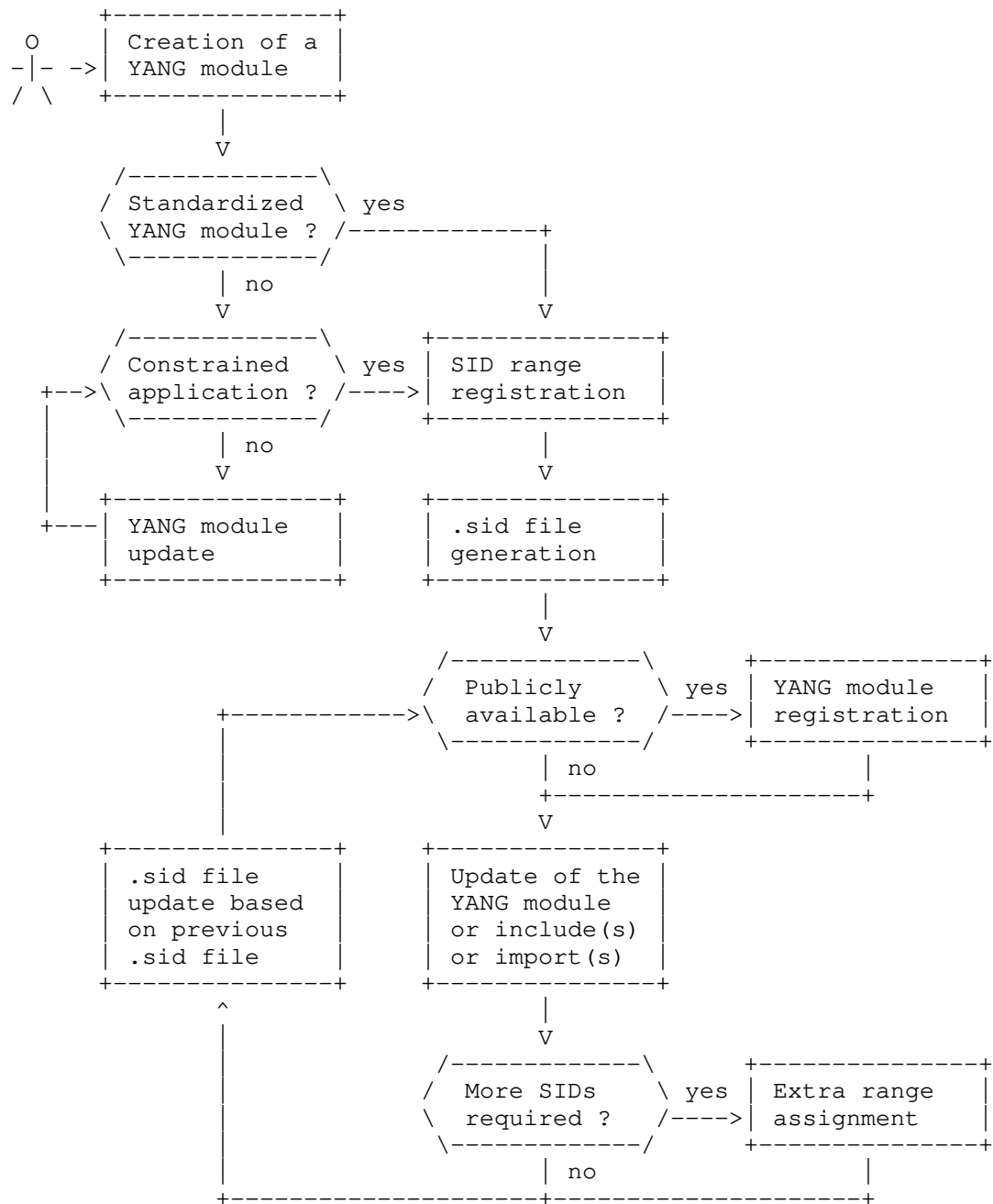
- o A tool extracts the different items defined for a specific YANG module.
- o The list of items is ordered by type and label.
- o SIDs are assigned sequentially for the entry point up to the size of the registered SID range. It is important to note that sequentially assigning SIDs optimizes the CBOR serialization due to the use of delta encoding.
- o If the number of items exceeds the SID range(s) allocated to a YANG module, an extra range is added for subsequent assignments.

SIDs are assigned permanently, items introduced by a new revision of a YANG module are added to the list of SIDs already assigned. This process can also be automated using the same method described above except that the assignment need to be restarted from the highest SID already assigned.

Section 5 defines a standard file format used to store and publish SIDs.

4. ".sid" file lifecycle

The following activity diagram summarize the life cycle of ".sid" files.



YANG modules are not necessarily created in the context of constrained applications. YANG modules can be implemented using NETCONF or RESTCONF without the need to assign SIDs.

As needed, authors of YANG modules can assign SIDs to their modules. This process starts by the registration of a SID range. Once a SID range is registered, the owner of this range assigns sub-ranges to each YANG module in order to generate the associated ".sid" files. Generation of ".sid" files SHOULD be performed using an automated tool.

Registration of the .sid file associated to a YANG module is optional but recommended to promote interoperability between devices and to avoid duplicate allocation of SIDs to a single YANG module.

Each time a YANG module or one of its imported module(s) or included sub-module(s) is updated, the ".sid" file MAY need to be updated. This update SHOULD also be performed using an automated tool.

If a new revision requires more SIDs than initially allocated, a new SID range MUST be added to the assignment ranges as defined in the ".sid" file header. These extra SIDs are used for subsequent assignments.

5. ".sid" file format

".sid" files are used to persist and publish SIDs assigned to the different YANG items of a specific YANG module. The following YANG module defined the structure of this file, encoding is performed using the rules defined in [I-D.ietf-netmod-yang-json].

```
<CODE BEGINS> file "ietf-sid-file@2015-12-16.yang"
module ietf-sid-file {
  namespace "urn:ietf:params:xml:ns:yang:ietf-sid-file";
  prefix sid;

  organization
    "IETF Core Working Group";

  contact
    "Ana Minaburo
    <ana@ackl.io>

    Alexander Pelov
    <mailto:a@ackl.io>

    Abhinav Somaraju
    <mailto:abhinav.somaraju@tridonic.com>
```

Laurent Toutain
<Laurent.Toutain@telecom-bretagne.eu>

Randy Turner
<mailto:Randy.Turner@landisgyr.com>

Michel Veillette
<mailto:michel.veillette@trilliantinc.com>;

description

"This module define the structure of the .sid files.
.sid files contains the identifiers (SIDs) assigned
to the different items defined in a YANG module.
SIDs are used to encode a data model defined in YANG
using CBOR.";

```
revision 2015-12-16 {  
  description  
    "Initial revision.";  
  reference  
    "RFC XXXX";  
  // RFC Ed.: replace XXXX with RFC number assigned to draft-ietf-core-yang-  
  cbor and remove this note  
}
```

```
typedef yang-identifier {  
  type string {  
    length "1..max";  
    pattern '[a-zA-Z_][a-zA-Z0-9\-\_\.]*';  
    pattern '\.|\.\.|\^[xX].*|\^[mM].*|\.\.\^[lL].*';  
  }  
  description  
    "A YANG identifier string as defined by the 'identifier'  
    rule in Section 12 of RFC 6020.";  
}
```

```
typedef revision-identifier {  
  type string {  
    pattern '\d{4}-\d{2}-\d{2}';  
  }  
  description  
    "Represents a date in YYYY-MM-DD format.";  
}
```

```
leaf module-name {  
  type yang-identifier;  
  description  
    "Name of the module associated with this .sid file.";  
}
```

```
leaf module-revision {
  type revision-identifier;
  description
    "Revision of the module associated with this .sid file.
    This leaf is not present if no revision statement is
    defined in the YANG module.";
}

list assignment-ranges {
  key "entry-point";
  description
    "Range(s) of SIDs available for assignment to the
    different items defined by the associated module.";

  leaf entry-point {
    type uint32;
    mandatory true;
    description
      "Lowest SID available for assignment.";
  }

  leaf size {
    type uint16;
    mandatory true;
    description
      "Number of SIDs available for assignment.";
  }
}

list items {
  key "type label";
  description
    "List of items defined by the associated YANG module.";

  leaf type {
    type string {
      pattern 'Module|Submodule|feature|' +
        'identity$|node$|notification$|rpc$|action$';
    }
    mandatory true;
    description
      "Item type assigned, this field can be set to:
      - 'Module'
      - 'Submodule'
      - 'feature'
      - 'identity'
      - 'node'
      - 'notification'
```

```
        - 'rpc'
        - 'action'";
    }

    leaf label {
        type string;
        mandatory true;
        description
            "Label associated to this item, can be set to:
            - a module name
            - a submodule name
            - a feature name
            - a base identity encoded as '/<base identity name>'
            - an identity encoded as '/<base identity name>/<identity name>'
            - a schema node path";
    }

    leaf sid {
        type uint32;
        mandatory true;
        description "Identifier assigned to this YANG item.";
    }
}
}
<CODE ENDS>
```

6. Security Considerations

The security considerations of [RFC7049] and [I-D.ietf-netmod-rfc6020bis] apply.

This document defines an new type of identifier used to encode data models defined in YANG [I-D.ietf-netmod-rfc6020bis]. As such, this identifier does not contribute to any new security issues in addition of those identified for the specific protocols or contexts for which it is used.

7. IANA Considerations

7.1. "SID" range registry

IANA is requested to create a registry for Structure Identifier (SID) ranges. This registry needs to guarantee that the ranges registered do not overlap. The registry SHALL record for each entry:

- o The entry point (first entry) of the registered SID range.
- o The size of the registered SID range.

- o The contact information of the owner of the range such as name, email address, and phone number.

The IANA policy for this registry is split into four tiers as follows:

- o The range of 0 to 999 and 0x40000000 to 0xFFFFFFFF are reserved for future extensions of this protocol. Allocation within these ranges require IETF review or IESG approval.
- o The range of 1000 to 59999 is reserved for standardized YANG modules. Allocation within this range requires publishing of the associated ".yang" and ".sid" files. (Specification required.)
- o The range of 60000 to 99999 is reserved for experimental YANG modules. Use of this range MUST NOT be used in operational deployments since these SIDs are not globally unique which limit their interoperability.
- o The range of 100000 to 0x3FFFFFFF is available on a first come first served basis. The only information required from the registrant is a valid contact information. The recommended size of the SID ranges allocated is 1,000 for private use and 10,000 for standard development organizations (SDOs). Registrants MAY request fewer or more SIDs based on their expected, sat needs. Allocation of a significantly larger SID range MAY required IETF review or IESG approval. IANA MAY delegate this registration process to one or multiple sub-registries. The recommended size of the SID range allocation for a sub-registry is 1,000,000.

Entry Point	Size	Registration Procedures
0	1,000	IETF review or IESG approval
1,000	59,000	Specification and associated ".yang" and ".sid" files required
60,000	40,000	Experimental use
100,000	0x3ffe7960	Contact information is required. Registration of the module name(s) and associated ".yang" and ".sid" files are optional.
0x40000000	2^64-0x40000000	Specification required, expert review

7.2. YANG module registry

Each registered SID range can be used to assign SIDs to one or more YANG modules. To track which YANG modules have been assigned and to avoid duplicate allocation, IANA is requested to provide a method to register and query the following information:

- o The YANG module name
- o The contact information of the author
- o The specification reference
- o The associated ".yang" file(s) (Optional)
- o The associated ".sid" file (Optional)

Registration of YANG modules is optional. When a YANG module is registered, the registrant MUST provide the module name and contact information and/or a specification reference.

The registration of the associated ".yang" and ".sid" files is optional. When provided, the validity of the files MUST be verified. This can be accomplished by a YANG validation tool specially modified to support ".sid" file verification. The SID range specified within the ".sid" file SHOULD also be checked against the "SID" range registry (Section 7.1) and against the other YANG modules registered to detect any duplicate use of SIDs.

Initial entries in this registry are as follows:

Entry Point	Size	Module name	Reference
1000	100	ietf-cool	[I-D.veillette-core-cool]
1100	400	iana-if-type	[RFC7224]
1500	100	ietf-interfaces	[RFC7223]
1600	100	ietf-ip	[RFC7277]
1700	100	ietf-system	[RFC7317]

8. Acknowledgments

The authors would like to thank Carsten Bormann for his help during the development of this document and his useful comments during the review process.

9. References

9.1. Normative References

- [I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language",
draft-ietf-netmod-rfc6020bis-14 (work in progress), June
2016.
- [I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG",
draft-ietf-netmod-yang-json-10 (work in progress), March
2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.

9.2. Informative References

- [I-D.veillette-core-cool]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A.
Minaburo, "Constrained Objects Language", draft-veillette-
core-cool-01 (work in progress), March 2016.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface
Management", RFC 7223, DOI 10.17487/RFC7223, May 2014,
<<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module",
RFC 7224, DOI 10.17487/RFC7224, May 2014,
<<http://www.rfc-editor.org/info/rfc7224>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management",
RFC 7277, DOI 10.17487/RFC7277, June 2014,
<<http://www.rfc-editor.org/info/rfc7277>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for
System Management", RFC 7317, DOI 10.17487/RFC7317, August
2014, <<http://www.rfc-editor.org/info/rfc7317>>.

Appendix A. ".sid" file example

The following .sid file (ietf-system@2014-08-06.sid) have been generated using the following yang modules:

- o ietf-system@2014-08-06.yang
- o ietf-yang-types@2013-07-15.yang
- o ietf-inet-types@2013-07-15.yang
- o ietf-netconf-acm@2012-02-22.yang
- o iana-crypt-hash@2014-04-04.yang

```
{
  "assignment-ranges": [
    {
      "entry-point": 1700,
      "size": 100
    }
  ],
  "module-name": "ietf-system",
  "module-revision": "2014-08-06",
  "items": [
    {
      "type": "Module",
      "label": "ietf-system",
      "sid": 1700
    },
    {
      "type": "feature",
      "label": "authentication",
      "sid": 1701
    },
    {
      "type": "feature",
      "label": "dns-udp-tcp-port",
      "sid": 1702
    },
    {
      "type": "feature",
      "label": "local-users",
      "sid": 1703
    },
    {
      "type": "feature",
      "label": "ntp",
```

```
    "sid": 1704
  },
  {
    "type": "feature",
    "label": "ntp-udp-port",
    "sid": 1705
  },
  {
    "type": "feature",
    "label": "radius",
    "sid": 1706
  },
  {
    "type": "feature",
    "label": "radius-authentication",
    "sid": 1707
  },
  {
    "type": "feature",
    "label": "timezone-name",
    "sid": 1708
  },
  {
    "type": "identity",
    "label": "/authentication-method",
    "sid": 1709
  },
  {
    "type": "identity",
    "label": "/authentication-method/local-users",
    "sid": 1710
  },
  {
    "type": "identity",
    "label": "/authentication-method/radius",
    "sid": 1711
  },
  {
    "type": "identity",
    "label": "/radius-authentication-type",
    "sid": 1712
  },
  {
    "type": "identity",
    "label": "/radius-authentication-type/radius-chap",
    "sid": 1713
  },
  {
```

```
    "type": "identity",
    "label": "/radius-authentication-type/radius-pap",
    "sid": 1714
  },
  {
    "type": "node",
    "label": "/system",
    "sid": 1715
  },
  {
    "type": "node",
    "label": "/system-state",
    "sid": 1716
  },
  {
    "type": "node",
    "label": "/system-state/clock",
    "sid": 1717
  },
  {
    "type": "node",
    "label": "/system-state/clock/boot-datetime",
    "sid": 1718
  },
  {
    "type": "node",
    "label": "/system-state/clock/current-datetime",
    "sid": 1719
  },
  {
    "type": "node",
    "label": "/system-state/platform",
    "sid": 1720
  },
  {
    "type": "node",
    "label": "/system-state/platform/machine",
    "sid": 1721
  },
  {
    "type": "node",
    "label": "/system-state/platform/os-name",
    "sid": 1722
  },
  {
    "type": "node",
    "label": "/system-state/platform/os-release",
    "sid": 1723
  }
```

```
,
{
  "type": "node",
  "label": "/system-state/platform/os-version",
  "sid": 1724
},
{
  "type": "node",
  "label": "/system/authentication",
  "sid": 1725
},
{
  "type": "node",
  "label": "/system/authentication/user",
  "sid": 1726
},
{
  "type": "node",
  "label": "/system/authentication/user-authentication-order",
  "sid": 1727
},
{
  "type": "node",
  "label": "/system/authentication/user/authorized-key",
  "sid": 1728
},
{
  "type": "node",
  "label": "/system/authentication/user/authorized-key/algorithm",
  "sid": 1729
},
{
  "type": "node",
  "label": "/system/authentication/user/authorized-key/key-data",
  "sid": 1730
},
{
  "type": "node",
  "label": "/system/authentication/user/authorized-key/name",
  "sid": 1731
},
{
  "type": "node",
  "label": "/system/authentication/user/name",
  "sid": 1732
},
{
  "type": "node",
```

```
    "label": "/system/authentication/user/password",
    "sid": 1733
  },
  {
    "type": "node",
    "label": "/system/clock",
    "sid": 1734
  },
  {
    "type": "node",
    "label": "/system/clock/timezone/timezone-name/timezone-name",
    "sid": 1735
  },
  {
    "type": "node",
    "label": "/system/clock/timezone/timezone-utc-offset/timezone-utc-offset",
    "sid": 1736
  },
  {
    "type": "node",
    "label": "/system/contact",
    "sid": 1737
  },
  {
    "type": "node",
    "label": "/system/dns-resolver",
    "sid": 1738
  },
  {
    "type": "node",
    "label": "/system/dns-resolver/options",
    "sid": 1739
  },
  {
    "type": "node",
    "label": "/system/dns-resolver/options/attempts",
    "sid": 1740
  },
  {
    "type": "node",
    "label": "/system/dns-resolver/options/timeout",
    "sid": 1741
  },
  {
    "type": "node",
    "label": "/system/dns-resolver/search",
    "sid": 1742
  },
}
```



```
{
  "type": "node",
  "label": "/system/dns-resolver/server",
  "sid": 1743
},
{
  "type": "node",
  "label": "/system/dns-resolver/server/name",
  "sid": 1744
},
{
  "type": "node",
  "label": "/system/dns-resolver/server/transport/udp-and-tcp/udp-and-tcp",
  "sid": 1745
},
{
  "type": "node",
  "label": "/system/dns-resolver/server/transport/udp-and-tcp/udp-and-tcp/ad
dress",
  "sid": 1746
},
{
  "type": "node",
  "label": "/system/dns-resolver/server/transport/udp-and-tcp/udp-and-tcp/po
rt",
  "sid": 1747
},
{
  "type": "node",
  "label": "/system/hostname",
  "sid": 1748
},
{
  "type": "node",
  "label": "/system/location",
  "sid": 1749
},
{
  "type": "node",
  "label": "/system/ntp",
  "sid": 1750
},
{
  "type": "node",
  "label": "/system/ntp/enabled",
  "sid": 1751
},
{
  "type": "node",
  "label": "/system/ntp/server",
```

```
    "sid": 1752
  },
  {
    "type": "node",
    "label": "/system/ntp/server/association-type",
    "sid": 1753
  },
  {
    "type": "node",
    "label": "/system/ntp/server/iburst",
    "sid": 1754
  },
  {
    "type": "node",
    "label": "/system/ntp/server/name",
    "sid": 1755
  },
  {
    "type": "node",
    "label": "/system/ntp/server/prefer",
    "sid": 1756
  },
  {
    "type": "node",
    "label": "/system/ntp/server/transport/udp/udp",
    "sid": 1757
  },
  {
    "type": "node",
    "label": "/system/ntp/server/transport/udp/udp/address",
    "sid": 1758
  },
  {
    "type": "node",
    "label": "/system/ntp/server/transport/udp/udp/port",
    "sid": 1759
  },
  {
    "type": "node",
    "label": "/system/radius",
    "sid": 1760
  },
  {
    "type": "node",
    "label": "/system/radius/options",
    "sid": 1761
  },
  {
```

```
    "type": "node",
    "label": "/system/radius/options/attempts",
    "sid": 1762
  },
  {
    "type": "node",
    "label": "/system/radius/options/timeout",
    "sid": 1763
  },
  {
    "type": "node",
    "label": "/system/radius/server",
    "sid": 1764
  },
  {
    "type": "node",
    "label": "/system/radius/server/authentication-type",
    "sid": 1765
  },
  {
    "type": "node",
    "label": "/system/radius/server/name",
    "sid": 1766
  },
  {
    "type": "node",
    "label": "/system/radius/server/transport/udp/udp",
    "sid": 1767
  },
  {
    "type": "node",
    "label": "/system/radius/server/transport/udp/udp/address",
    "sid": 1768
  },
  {
    "type": "node",
    "label": "/system/radius/server/transport/udp/udp/authentication-port",
    "sid": 1769
  },
  {
    "type": "node",
    "label": "/system/radius/server/transport/udp/udp/shared-secret",
    "sid": 1770
  },
  {
    "type": "rpc",
    "label": "/set-current-datetime",
    "sid": 1771
  }
```

```
    },  
    {  
      "type": "rpc",  
      "label": "/set-current-datetime/input/current-datetime",  
      "sid": 1772  
    },  
    {  
      "type": "rpc",  
      "label": "/system-restart",  
      "sid": 1773  
    },  
    {  
      "type": "rpc",  
      "label": "/system-shutdown",  
      "sid": 1774  
    }  
  ]  
}
```

Authors' Addresses

Abhinav Somaraju (editor)
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: <http://www.landisgyr.com/>

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

CoRE
Internet Draft
Intended status: Informational
Expires: November 2016

A. Bhattacharyya
S. Bandyopadhyay
A. Pal
T. Bose
Tata Consultancy Services Ltd.
May 12, 2016

CoAP option for no server-response
draft-tcs-coap-no-response-option-17

Abstract

There can be M2M scenarios where responses from a server against requests from client are redundant. This kind of open-loop exchange (with no response path from the server to the client) may be desired to minimize resource consumption in constrained systems while updating a bulk of resources simultaneously, or updating a resource with a very high frequency. CoAP already provides Non-confirmable (NON) messages that are not acknowledged by the recipient. However, the request/response semantics still require the server to respond with a status code indicating "the result of the attempt to understand and satisfy the request".

This specification introduces a CoAP option called 'No-Response'. Using this option the client can explicitly express to the server its disinterest in all responses against the particular request. This option also provides granular control to enable expression of disinterest to a particular class of response or a combination of response-classes. The server MAY decide to suppress the response by not transmitting it back to the client according to the value of No-Response option in the request. This option may be effective for both unicast and multicast requests. This document also discusses a few exemplary applications which benefit from this option.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on November 12, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction.....	3
1.1. Potential Benefits.....	3
1.2. Terminology.....	4
2. Option Definition.....	4
2.1. Granular Control over Response Suppression.....	5
2.2. Method-specific Applicability Consideration.....	7
3. Miscellaneous Aspects.....	8
3.1. Re-using Tokens.....	9
3.2. Taking Care of Congestion Control and Server-side Flow Control.....	10
3.3. Considerations Regarding Caching of Responses.....	11
3.4. Handling No-Response Option for a HTTP-to-CoAP Reverse Proxy.....	11
4. Exemplary Application Scenarios.....	11
4.1. Frequent Update of Geo-location from Vehicles to Backend Server.....	11

4.1.1. Using No-Response with PUT.....	13
4.1.2. Using No-Response with POST.....	13
4.1.2.1. POST updating a fixed target resource.....	13
4.1.2.2. POST updating through query-string.....	14
4.2. Multicasting Actuation Command from a Handheld Device to a Group of Appliances.....	15
4.2.1. Using Granular Response Suppression.....	16
5. IANA Considerations.....	16
6. Security Considerations.....	16
7. Acknowledgments.....	16
8. References.....	16
8.1. Normative References.....	16
8.2. Informative References.....	17

1. Introduction

This specification defines a new option for Constrained Application Protocol (CoAP) [RFC7252] called 'No-Response'. This option enables clients to explicitly express their disinterests in receiving responses back from the server. The disinterest can be expressed at the granularity of response classes (e.g., 2.xx or the combination of 2.xx and 5.xx). By default this option indicates interest in all response classes. The server MAY decide to suppress the response by not transmitting it back to the client according to the value of the No-Response option in the request.

Along with the technical details this document presents some practical application scenarios which bring out the usefulness of this option.

Wherever, in this document, it is mentioned that a request from a client is with No-Response the intended meaning is that the client expresses its disinterest for all or some selected classes of responses.

1.1. Potential Benefits

Use of No-Response option should be driven by typical application requirement and, particularly, characteristics of the information to be updated. If this option is opportunistically used in a fitting M2M application then the concerned system may benefit in the following aspects (however, it is to be noted, this option is elective and servers can simply ignore the preference expressed by the client):

* Reduction in network congestion due to effective reduction of the overall traffic.

* Reduction in server-side load by relieving the server from responding to each request when not necessary.

* Reduction in battery consumption at the constrained end-point(s).

* Reduction in overall communication cost.

1.2. Terminology

The terms used in this document are in conformance with those defined in [RFC7252].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119.

2. Option Definition

The properties of No-Response option are given in Table 1.

Number	C	U	N	R	Name	Format	Length	Default
258		X	-		No-Response	uint	0-1	0

Table 1: Option Properties

This option is a request option. It is Elective and Non-Repeatable. This option is Unsafe-to-forward as the intermediary MUST know how to interpret this option. Otherwise the intermediary, without knowledge about the special unidirectional nature of the request, would wait for responses.

Note: Since CoAP maintains a clear separation between the request/response and the message sub-layer, this option does not have any dependency on the type of message (Confirmable/Non-confirmable). So, even the absence of message sub-layer (ex. CoAP-over-TCP [I-D.ietf-core-coap-tcp-tls-01]) should have no effect on the interpretation of this option. However, considering the CoAP-over-UDP scenario [RFC7252], NON type of messages are best fitting with this option, considering the expected benefits out of it. Using No-Response with NON messages gets rid of any

kind of reverse traffic and the interaction becomes completely open-loop.

Using this option with CON type of requests may not serve the desired purpose if piggybacked responses are triggered. But, in case the server responds with a separate response (which, perhaps, the client does not care about) then this option can be useful. Suppressing the separate response reduces traffic by one additional CoAP message in this case.

This option contains values to indicate disinterest in all or a particular class or combination of classes of responses as described in the next sub-section.

2.1. Granular Control over Response Suppression

This option enables granular control over response suppression by allowing the client to express its disinterest in a typical class or combination of classes of responses. For example, a client may explicitly tell the receiver that no response is required unless something 'bad' happens and a response of class 4.xx or 5.xx is to be fed back to the client. No response of the class 2.xx is required in such case.

Note: Section 2.7 of [RFC7390] describes a scheme where a server in the multicast group may decide on its own to suppress responses for group communication with granular control. The client does not have any knowledge about that. However, on the other hand, the 'No-Response' option enables the clients to explicitly inform the servers about its disinterest in responses. Such explicit control on the client side may be helpful for debugging network resources. An example scenario is described in Section 4.2.1.

The server MUST send back responses of the classes for which the client has not expressed any dis-interest. There may be instances where a server, on its own, decides to suppress responses. An example is suppression of responses by multicast servers as described in Section 2.7 of [RFC7390]. If such a server receives a request with a No-Response option showing 'interest' in specific response classes (i.e., not expressing disinterest for these options), then any default behaviour of suppressing response, if present, MUST be overridden to deliver those responses which are of interest to the client.

So, for example, suppose a multicast server suppresses all responses by default and receives a request with a No-Response option expressing disinterest in 2.xx (success) responses only. Note that

the option value naturally expresses interest in error responses 4.xx/5.xx in this case. Then the server must send back a response if the concerned request caused an error.

The option value is defined as a bit-map (Table 2) to achieve granular suppression. Its length can be 0 byte (empty value) or 1 byte.

Value	Binary Representation	Description
0	<empty>	Interested in all responses.
2	00000010	Not interested in 2.xx responses.
8	00001000	Not interested in 4.xx responses.
16	00010000	Not interested in 5.xx responses.

Table 2: Option values

The conventions used in deciding the option values are:

1. To suppress an individual class: Set bit number (n-1) starting from the LSB (bit number 0) to suppress all responses belonging to class n.xx. So,

$$\text{option value to suppress n.xx class} = 2^{*(n-1)}.$$

2. To suppress combination of classes: Set each corresponding bit according to point 1 above. Example: The option value will be 18 (binary: 00010010) to suppress both 2.xx and 5.xx responses. This is essentially bitwise OR of the corresponding individual values for suppressing 2.xx and 5.xx. The "CoAP Response Codes" registry (Ref. Section 12.1.2 of [RFC7252]) defines 2.xx, 4.xx and 5.xx responses. So, an option value of 26 (binary: 00011010) will request to suppress all response codes defined in [RFC7252].

Note: When No-Response is used with value 26 in a request the client end-point SHOULD cease listening to response(s) against the particular request. On the other hand, showing interest in at

least one class of response means that the client end-point can no longer completely cease listening activity and must be configured to listen up to some application specific time-out period for the particular request. The client end-point never knows whether the present request will be a success or a failure. Thus, for example, if the client decides to open up the response for errors (4.xx and 5.xx) then it has to wait for the entire time-out period even for the instances where the request is successful (and the server is not supposed to send back a response). A point to be noted in this context is that there may be situations when the response on errors might get lost. In such a situation the client would wait up to the time-out period but will not receive any response. But this should not lead to the impression to the client that the request was necessarily successful. In other words, in this case the client cannot distinguish between response suppression and message loss. The application designer needs to tackle such situation. For example, while performing frequent updates, the client may strategically interweave requests without No-Response option into a series of requests with No-Response to check time to time if things are fine at the server end and the server is actively responding.

2.2. Method-specific Applicability Consideration

The following table provides a ready-reference on the possible applicability of this option for all the four REST methods. This table is prepared in view of the type of possible interactions foreseen at time of preparing this specification. Capitalization of key words like "SHOULD NOT", etc. have not been deliberately used in this table as this table is only suggestive.

Method Name	Remarks on applicability
GET	This should not be used with conventional GET request when the client requests the contents of a resource. However, this option may be useful for exceptional cases where GET requests has side effects. For instance, the proactive 'cancellation' procedure for observing request [RFC7641] requires a client to issue a GET request with Observe option set to 1 ('deregister'). In case it is more efficient to use this deregistration instead of reactive cancellation (rejecting the next notification with RST), the client MAY express its disinterest in the response to such a GET request.
PUT	Suitable for frequent updates (particularly in NON messages) on existing resources. Might not be useful when PUT is used to create a new resource as it may be important for the client to know that the resource creation was actually successful in order to carry out future actions. Also, it may be important to ensure that a resource was actually created rather than updating an existing resource.
POST	If POST is used to update a target resource then No-Response can be used in the same manner as in PUT. This option may also be useful while updating through query strings rather than updating a fixed target resource (see Section 4.1.2.2 for an example).
DELETE	Deletion is usually a permanent action and if the client likes to ensure that the deletion request was properly executed then this option should not be used with the request.

Table 3: Suggested applicability of No-Response for different REST methods

3. Miscellaneous Aspects

This section further describes important implementation aspects worth considering while using the No-Response option. The following discussion contains guidelines and requirements (derived by combining [RFC7252], [RFC7390] and [RFC5405]) for the application developer.

3.1. Re-using Tokens

Tokens provide a matching criteria between a request and the corresponding response. The life of a Token starts when it is assigned to a request and ends when the final matching response is received. Then the Token can again be re-used. However, a request with No-Response typically does not have any guaranteed response path. So, the client has to decide on its own about when it can retire a Token which has been used in an earlier request so that the Token can be reused in a future request. Since the No-Response option is 'elective', a server which has not implemented this option will respond back. This leads to the following two scenarios:

The first scenario is, the client is never going to care about any response coming back or about relating the response to the original request. In that case it MAY reuse the Token value at liberty.

However, as a second scenario, let us consider that the client sends two requests where the first request is with No-Response and the second request, with same Token, is without No-Response. In this case a delayed response to the first one can be interpreted as a response to the second request (client needs a response in the second case) if the time interval between using the same Token is not long enough. This creates a problem in the request-response semantics.

The most ideal solution would be to always use a unique Token for requests with No-Response. But if a client wants to reuse a Token then in most practical cases the client implementation SHOULD implement an application specific reuse time after which it can reuse the Token. A minimum reuse time for Tokens with a similar expression as in Section 2.5 of [RFC7390] SHOULD be used:

$$\text{TOKEN_REUSE_TIME} = \text{NON_LIFETIME} + \text{MAX_SERVER_RESPONSE_DELAY} + \text{MAX_LATENCY}.$$

NON_LIFETIME and MAX_LATENCY are defined in 4.8.2 of [RFC7252]. MAX_SERVER_RESPONSE_DELAY has same interpretation as in Section 2.5 of [RFC7390] for multicast request. For a unicast request, since the message is sent to only one server, MAX_SERVER_RESPONSE_DELAY means the expected maximum response delay from the particular server to which client sent the request. For multicast requests, MAX_SERVER_RESPONSE_DELAY has the same interpretation as in Section 2.5 of [RFC7390]. So, for multicast it is the expected maximum server response delay "over all servers that the client can send a multicast request to". This response delay for a given server includes its specific Leisure period; where Leisure is defined in

Section 8.2 of [RFC7252]. In general, the Leisure for a server may not be known to the client. A lower bound for Leisure, `lb_Leisure`, is defined in [RFC7252], but not an upper bound as is needed in this case. Therefore the upper bound can be estimated by taking N ($N \gg 1$) times the lower bound `lb_Leisure`:

$$\text{lb_Leisure} = S * G / R$$

(S = estimated response size; R = data transfer rate; G = group size estimate)

Any estimate of `MAX_SERVER_RESPONSE_DELAY` MUST be larger than `DEFAULT_LEISURE` as defined in [RFC7252].

Note: If it is not possible for the client to get a reasonable estimate of the `MAX_SERVER_RESPONSE_DELAY` then the client, to be safe, SHOULD use a unique Token for each stream of message.

3.2. Taking Care of Congestion Control and Server-side Flow Control

This section provides guidelines for basic congestion control. Better congestion control mechanisms can be designed as future work.

If this option is used with NON messages then the interaction becomes completely open-loop. Absence of any feedback from the server-end affects congestion-control mechanism. In this case the communication pattern maps to the scenario where the application cannot maintain an RTT estimate as described in Section 3.1.2 of [RFC5405]. Hence, following [RFC5405], a 3 seconds interval is suggested as the minimum interval between successive updates and use even less aggressive rate when possible. However, in case of more frequent update rates the application MUST have some knowledge about the channel and an application developer MUST interweave occasional closed-loop exchanges (e.g. NON messages without No-Response or CON messages) to get an RTT estimate between the endpoints.

Interweaving requests without No-Response is a MUST in case of aggressive request rate for applications where server-side flow control is necessary. For example, as proposed in [I-D.koster-core-coap-pubsub], a broker MAY return "4.29 Too Many Requests" in order to request a client to slow down the request rate. Interweaving requests without No-Response allows the client to listen to such response.

3.3. Considerations Regarding Caching of Responses

The cacheability of CoAP responses does not depend on the request method, but it depends on the Response Code. The No-Response option does not lead to any impact on cacheability of responses. If a request containing No-Response triggers a cacheable response then the response **MUST** be cached. However, the response **MAY** not be transmitted considering the value of the No-Response option in the request.

For example, if a request with No-Response triggers a cacheable response of 4.xx class with Max-Age !=0 then the response must be cached. The cache will return the response to subsequent similar requests without No-Response as long as the Max-Age is not elapsed.

3.4. Handling No-Response Option for a HTTP-to-CoAP Reverse Proxy

A HTTP-to-CoAP reverse proxy **MAY** translate an incoming HTTP request to a corresponding CoAP request indicating that no response is required (showing disinterest in all classes of responses) based on some application specific requirement. In this case it is **RECOMMENDED** that the reverse proxy generates an HTTP response with status code 204 (No Content) when such response is allowed. The generated response is sent after the proxy has successfully sent out the CoAP request.

In case the reverse proxy applies No-Response for particular class(es) of response(s) it will wait for responses up to an application specific maximum time (T_{max}) before responding back to the HTTP-side. If a response of a desired class is received within T_{max} then the response gets translated to HTTP as defined in [I-D.ietf-core-http-mapping]. However if the proxy does not receive any response within T_{max} , it is **RECOMMENDED** that the reverse Proxy sends an HTTP response with status code 204 (No Content) when allowed for the specific HTTP request method.

4. Exemplary Application Scenarios

This section describes some exemplary application scenarios which may potentially benefit from the use of No-Response option.

4.1. Frequent Update of Geo-location from Vehicles to Backend Server

Let us consider an intelligent traffic system (ITS) consisting of vehicles equipped with a sensor-gateway comprising sensors like GPS and Accelerometer. The sensor-gateway acts as a CoAP client. It connects to the Internet using a low-bandwidth cellular (e.g. GPRS)

connection. The GPS co-ordinates of the vehicle are periodically updated to the backend server.

While performing frequent location update, retransmitting (through the CoAP CON mechanism) a location co-ordinate which the vehicle has already left in the meantime is not efficient as it adds redundant traffic to the network. Therefore, the updates are done using NON messages. However, given the huge number of vehicles updating frequently, the NON exchange will also trigger huge number of responses from the backend. Thus the cumulative load on the network will be quite significant. Also, the client in this case may not be interested in getting responses against location update request for the location it has already crossed in the meantime and a next location update is imminent.

On the contrary, if the client end-points on the vehicles explicitly declare that they do not need any status response back from the server then load will be reduced significantly. The assumption is that, since the update rate is high, stray losses in geo-location reports will be compensated with the large update rate.

Note: It may be argued that the above example application can also be implemented using Observe option ([RFC7641]) with NON notifications. But, in practice, implementing with Observe would require lot of book-keeping at the data-collection end-point at the backend (observer). The observer needs to maintain all the observe relationships with each vehicle. The data collection end-point may be unable to know all its data sources beforehand. The client end-points at vehicles may go offline or come back online randomly. In case of Observe the onus is always on the data collection end-point to establish an observe relationship with each data-source. On the other hand, implementation will be much simpler if the initiative is left on the data-source to carry out updates using No-Response option. Another way of looking at it is, the implementation choice depends on the perspective of interest to initiate the update. In an Observe scenario the interest is expressed by the data-consumer. On the contrary, the classic update case applies when the interest is from the data-producer. The 'No-Response' option enables to make classic updates further less resource consuming.

Following subsections illustrate some exemplary exchanges based on the application described above.

4.1.1.1. Using No-Response with PUT

Each vehicle is assigned a dedicated resource "vehicle-stat-<n>", where <n> can be any string uniquely identifying the vehicle. The update requests are sent over NON type of messages. The No-Response option causes the server not to respond back.

Client	Server
+-----> PUT	Header: PUT (T=NON, Code=0.03, MID=0x7d38) Token: 0x53 Uri-Path: "vehicle-stat-00" Content Type: text/plain No-Response: 26 Payload: "VehID=00&RouteID=DN47&Lat=22.5658745&Long=88.4107966667& Time=2013-01-13T11:24:31"
	[No response from the server. Next update in 20s.]
+-----> PUT	Header: PUT (T=NON, Code=0.03, MID=0x7d39) Token: 0x54 Uri-Path: "vehicle-stat-00" Content Type: text/plain No-Response: 26 Payload: "VehID=00&RouteID=DN47&Lat=22.5649015&Long=88.4103511667& Time=2013-01-13T11:24:51"

Figure 1: Exemplary unreliable update with No-Response option using PUT.

4.1.1.2. Using No-Response with POST

4.1.1.2.1. POST updating a fixed target resource

In this case POST acts the same way as PUT. The exchanges are same as above. The updated values are carried as payload of POST as shown in Figure 2.

Client	Server
+-----> POST	Header: POST (T=NON, Code=0.02, MID=0x7d38) Token: 0x53 Uri-Path: "vehicle-stat-00" Content Type: text/plain No-Response: 26 Payload: "VehID=00&RouteID=DN47&Lat=22.5658745&Long=88.4107966667& Time=2013-01-13T11:24:31"
	[No response from the server. Next update in 20s.]
+-----> POST	Header: POST (T=NON, Code=0.02, MID=0x7d39) Token: 0x54 Uri-Path: "vehicle-stat-00" Content Type: text/plain No-Response: 26 Payload: "VehID=00&RouteID=DN47&Lat=22.5649015&Long=88.4103511667& Time=2013-01-13T11:24:51"

Figure 2: Exemplary unreliable update with No-Response option using POST as the update-method.

4.1.2.2. POST updating through query-string

It may be possible that the backend infrastructure deploys a dedicated database to store the location updates. In such a case the client can update through a POST by sending a query string in the URI. The query-string contains the name/value pairs for each update. 'No-Response' can be used in same manner as for updating fixed resources. The scenario is depicted in Figure 3.

Client	Server
<pre>+-----> POST</pre>	<pre>Header: POST (T=NON, Code=0.02, MID=0x7d38) Token: 0x53 Uri-Path: "updateOrInsertInfo" Uri-Query: "VehID=00" Uri-Query: "RouteID=DN47" Uri-Query: "Lat=22.5658745" Uri-Query: "Long=88.4107966667" Uri-Query: "Time=2013-01-13T11:24:31" No-Response: 26</pre>
	<pre>[No response from the server. Next update in 20 secs.]</pre>
<pre>+-----> POST</pre>	<pre>Header: POST (T=NON, Code=0.02, MID=0x7d39) Token: 0x54 Uri-Path: "updateOrInsertInfo" Uri-Query: "VehID=00" Uri-Query: "RouteID=DN47" Uri-Query: "Lat=22.5649015" Uri-Query: "Long=88.4103511667" Uri-Query: "Time=2013-01-13T11:24:51" No-Response: 26</pre>

Figure 3: Exemplary unreliable update with No-Response option using POST with a query-string to insert update information to backend database.

4.2. Multicasting Actuation Command from a Handheld Device to a Group of Appliances

A handheld device (e.g. a smart phone) may be programmed to act as an IP enabled switch to remotely operate on a single or group of IP enabled appliances. For example, a multicast request to switch on/off all the lights of a building can be sent. In this case the IP switch application can use the No-Response option in a NON request message to reduce the traffic generated due to simultaneous CoAP responses from all the lights.

Thus No-Response helps in reducing overall communication cost and the probability of network congestion in this case.

4.2.1. Using Granular Response Suppression

The IP switch application may optionally use granular response suppression such that the error responses are not suppressed. In that case the lights which could not execute the request would respond back and be readily identified. Thus, explicit suppression of option classes by the multicast client may be useful to debug the network and the application.

5. IANA Considerations

The IANA has previously assigned number 284 to this option in the CoAP Option Numbers Registry. IANA is requested to change this as below:

Number	Name	Reference
258	No-Response	Section 2 of this document

6. Security Considerations

The No-Response option defined in this document presents no security considerations beyond those in Section 11 of the base CoAP specification [RFC7252].

7. Acknowledgments

Thanks to Carsten Bormann, Matthias Kovatsch, Esko Dijk, Bert Greevenbosch, Akbar Rahman and Klaus Hartke for their valuable inputs.

8. References

8.1. Normative References

[RFC7252]

Shelby, Z., Hartke, K. and Bormann, C., "Constrained Application Protocol (CoAP)", RFC 7252, June, 2014

8.2. Informative References

[RFC7641]

Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, September, 2015

[RFC7390]

Rahman, A. and Dijk, E., "Group Communication for CoAP", RFC 7390, October, 2014

[RFC5405]

Eggert, L. and Fairhurst, G., "Unicast UDP Usage Guidelines for Application Designers", RFC 5405, November, 2008

[I-D.ietf-core-http-mapping]

Castellani, A., et al., "Guidelines for HTTP-CoAP Mapping Implementations", draft-ietf-core-http-mapping-09, April 6, 2016

[I-D.koster-core-coap-pubsub]

Koster, M., et al., "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-04, November 5, 2015

[I-D.ietf-core-coap-tcp-tls-01]

Bormann, C., et al., "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-tcp-tls-01, November 19, 2015

[Mobiquitous 2013]

Bhattacharyya, A., Bandyopadhyay, S. and Pal, A., "ITS-light: Adaptive lightweight scheme to resource optimize intelligent transportation tracking system (ITS)-Customizing CoAP for opportunistic optimization", 10th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous 2013), December, 2013.

[Sensys 2013]

Bandyopadhyay, S., Bhattacharyya, A. and Pal, A., "Adapting protocol characteristics of CoAP using sensed indication for vehicular

analytics", 11th ACM Conference on Embedded Networked Sensor Systems (Sensys 2013), November, 2013.

Authors' Addresses

Abhijan Bhattacharyya
Tata Consultancy Services Ltd.
Kolkata, India

Email: abhijan.bhattacharyya@tcs.com

Soma Bandyopadhyay
Tata Consultancy Services Ltd.
Kolkata, India

Email: soma.bandyopadhyay@tcs.com

Arpan Pal
Tata Consultancy Services Ltd.
Kolkata, India

Email: arpan.pal@tcs.com

Tulika Bose
Tata Consultancy Services Ltd.
Kolkata, India

Email: tulika.bose@tcs.com

CoRE
Internet-Draft
Intended status: Best Current Practice
Expires: September 12, 2016

R. Turner, Ed.
Landis+Gyr
M. Veillette
Trilliant Networks Inc.
A. Pelov
Acklio
A. Somaraju
Tridonic GmbH & Co KG
A. Minaburo
Acklio
March 11, 2016

CoOL Problem Statement
draft-turner-core-cool-problem-statement-00

Abstract

A significant part of the next tens of billion of devices that will be connected to the Internet will be constrained devices, connecting over constrained networks. Managing these devices and the networks they form in a consistent, scalable, extensible, secure, energy-efficient manner with low computational and protocol complexity cannot be done in an ad-hoc manner. This document outlines the problem at hand and provides a roadmap of the possible solutions developed at the IETF. The description includes the basic constrained management problem, as well as properties of the solution. Details as to the Constrained Objects Language (CoOL) protocol itself can be found in companion documents.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Problem Statement	2
3. Why CoOL?	4
4. Roadmap	4
5. Security Considerations	5
Authors' Addresses	6

1. Introduction

The need exists for a unified approach to network management of constrained devices as well as constrained networks. Constrained devices imply the solution should require minimal resources (CPU, memory) from the device platform; constrained networks imply the network management solution should impose minimal traffic on the network to accomplish a particular management objective.

2. Problem Statement

The problem of network management for constrained networks and devices includes a number of requirements not necessarily addressed by existing solutions. Any solution must be conservative with "when" network traffic is required, as well as "how much" traffic is required in order to fulfill network management functions. In addition, a solution should support "traditional" network management functions that have been useful in a variety of legacy use-cases, as well as new functionalities that address the evolving constrained device and constrained device scenarios. In this context, the term "constrained" implies limited resources (RAM, FLASH), as well as limited CPU resources. Therefore, the amount of code necessary to achieve network management functionality will need to be as small as possible, as well as the amount of RAM necessary to achieve the

functionality will be limited. Likewise, minimal network bandwidth should be required to support a solution.

An ideal solution SHOULD therefore possess the following characteristics:

- o Simple and efficient
 - * Low memory footprint (RAM)
 - * Low binary footprint (flash)
 - * Low protocol complexity (simple state machine)
- o Scalable
 - * Thing-to-thing management
 - * Zero feature negotiation required
- o Compact on the air/wire
- o Secure
- o Extensible
- o Interoperable

The IETF is coalescing around a set of solutions for transport of applications on constrained networks (CoAP). Since constrained devices will likely include support for this constrained "stack", it would be advantageous to reuse this constrained device stack for network management as well, to address the constrained device property of limited resources.

Additionally, the IETF is moving towards YANG as a data modeling language for configuration and state data often attributed to network management problems. Therefore, any constrained device/network management solution should attempt to reuse this information when and where possible.

YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol (RFC 6241), NETCONF remote procedure calls, and NETCONF notifications. YANG was originally designed to work with the NETCONF configuration protocol; however, the idea of constrained networks and devices was not a factor in the design of NETCONF/YANG. Any solution that attempts to use YANG in a constrained environment should consider constrained

device and networking properties to the application of YANG in these scenarios.

It would be advantageous to model the particular constrained network management functionality on the evolving NETCONF/RESTCONF operations, since some level of semantic interoperability might be expected by management systems that mix constrained and non-constrained management domains.

One design element that could reduce the amount of traffic "on the wire" is requiring less metadata in management transactions. Instead of endpoints semantically parsing the meaning of the data and/or traffic, the knowledge of the data and how it is expected to be used is, instead, required on the endpoints, a priori.

3. Why CoOL?

Currently proposed solutions for constrained management do not specifically address the requirements previously suggested in this memo. The solution introduced by CoOL seeks to remedy this by introducing an alternative method for operations and encoding "on the wire". CoOL will address these requirements while still utilizing the IETF application "stack" and management data modeling language (YANG). The alternative method employed by CoOL utilizes a more concise representation of management transactions (specifically management "data").

The evolution of the CoOL solution will recognize the proliferation of the "pub/sub" design pattern by relying on extensions both at the CoAP, as well as CoOL protocol layers where needed. Incorporating the pub/sub design pattern will assist in the application of CoOL into larger scale networks (both constrained and non-constrained).

4. Roadmap

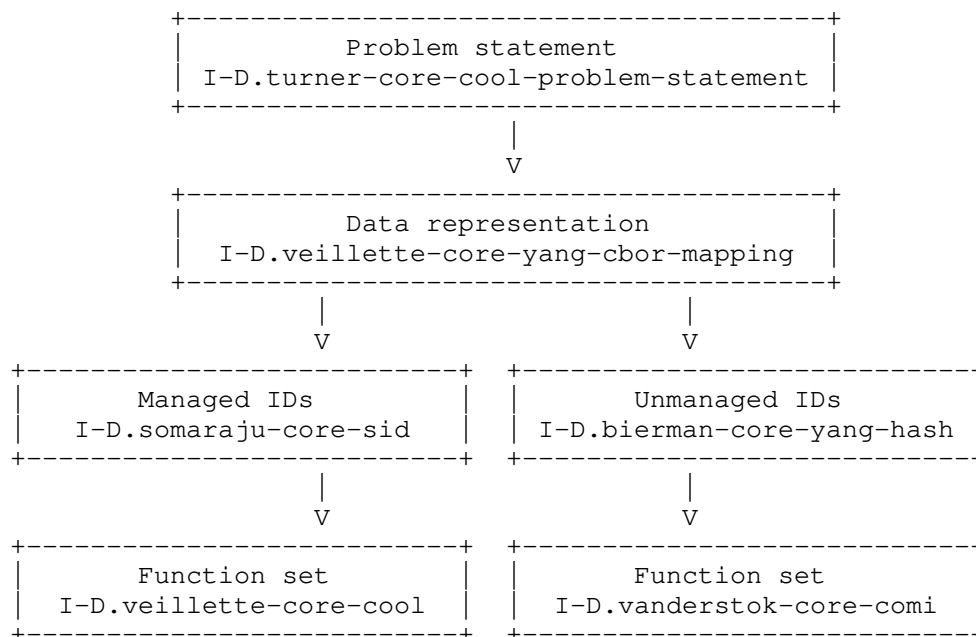
Within the CORE group, there are currently two threads of development of a management interface for constrained devices and networks. They both converge on the common grounds of using CoAP as fundamental application protocol. Both solutions bring the YANG modeling language to the world of constrained networks and devices, thus bridging the gap to core network management.

These two approaches differ by their runtime-vs-compile time complexity and efficiency. The first approach is based on unmanaged identifiers (YANG hashes), which require zero compile-time efforts in exchange for decreased runtime efficiency. The second approach is based on managed identifiers (SID), which takes the opposite

direction, requiring more upfront preparation, aiming at maximal efficiency, deterministic behavior and improved scalability.

Both solutions converge around the YANG data representation expressed in CBOR, as well as the common comprehension of the problem statement. The managed and unmanaged identifier spaces have their specific underlying function sets. These function set drafts define the well-known REST resources that provide the device management services.

The following drafts are currently available:



5. Security Considerations

The security considerations applicable to network management of enterprise networks is similar to, but different than that of constrained networks given the potential risk involved. The risk involved to enterprise networks could be local to an organization (assets, reputation). However, in the case of constrained networks, it is reasonable to assume significant risk due to the types of application domains constrained devices would be applied to (sensors controlling everything from home automation to medical devices). With this risk should come a more strict understanding of the attack vectors and vulnerabilities of any and all protocols in use in constrained networks, especially those protocols tasked with the

management of a device. The CoOL working group will attempt to reuse applicable ideas and technology originating from other IETF working groups to address the problem of security. The initial focus of security will involve the integrity and trustworthiness of information originating from CoOL managed endpoints. The confidentiality of this information will also be considered.

Authors' Addresses

Randy Turner (editor)
Landis+Gyr
30000 Mill Creek Ave
Alpharetta, Georgia 30022
USA

Phone: 678-258-1292
Email: randy.turner@landisgyr.com

Michel Veillette
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

core
Internet-Draft
Intended status: Standards Track
Expires: September 8, 2016

P. van der Stok
consultant
A. Bierman
YumaWorks
March 7, 2016

CoAP Management Interface
draft-vanderstok-core-comi-09

Abstract

This document describes a network management interface for constrained devices, called CoMI. CoMI is an adaptation of the RESTCONF protocol for use in constrained devices and networks. The Constrained Application Protocol (CoAP) is used to access management data resources specified in YANG, or SMIV2 converted to YANG. CoMI use the YANG to CBOR mapping and encodes YANG names to reduce payload size.

Note

Discussion and suggestions for improvement are requested, and should be sent to core@ietf.org.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 8, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
1.1.1. Tree Diagrams	5
2. CoMI Architecture	5
2.1. RESTCONF/YANG Architecture	9
2.1.1. Major differences between RESTCONF and CoMI	9
2.2. Compression of data-node instance identifier	10
3. CoAP Interface	11
4. MG Function Set	13
4.1. Data Retrieval	13
4.1.1. GET	13
4.1.2. Using the 'select' Parameter	14
4.1.3. Using the 'content' query parameter	14
4.1.4. Retrieval Examples	15
4.2. Data Editing	22
4.2.1. Data Ordering	22
4.2.2. POST	22
4.2.3. PUT	23
4.2.4. PATCH	23
4.2.5. DELETE	27
4.2.6. Editing Multiple Resources	27
4.3. Notify functions	28
4.4. RPC statements	30
4.4.1. Encoding RPC input parameters	31
4.4.2. Encoding RPC output parameters	32
4.5. Use of Block	32
4.6. Resource Discovery	33
4.7. Error Return Codes	35
5. Error Handling	36
6. Security Considerations	38
7. IANA Considerations	38
8. Acknowledgements	39
9. Changelog	39
10. References	42
10.1. Normative References	42
10.2. Informative References	44

Appendix A. Payload and Server sizes	46
Appendix B. Comparison with LWM2M	47
Authors' Addresses	48

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. The messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

This draft describes the CoAP Management Interface which uses CoAP methods to access structured data defined in YANG [RFC6020]. This draft is complementary to the draft [I-D.ietf-netconf-restconf] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data sets, specified in a standardized language such as YANG, promotes interoperability between devices and applications from different manufacturers. A large amount of Management Information Base (MIB) [RFC3418] [mibreg] specifications already exists for monitoring purposes. This data can be accessed in RESTCONF or CoMI if the server converts the SMIV2 modules to YANG, using the mapping rules defined in [RFC6643].

RESTCONF allows access to data resources contained in NETCONF [RFC6241] data-stores. RESTCONF messages can be encoded in XML [XML] or JSON [RFC7159]. The GET method is used to retrieve data resources and the POST, PUT, PATCH, and DELETE methods are used to create, replace, merge, and delete data resources.

CoMI supports the methods GET, PUT, PATCH, POST and DELETE. The payload of CoMI is encoded in CBOR [RFC7049] which can be automatically generated from JSON [RFC7159]. CBOR has a binary format and hence has more coding efficiency than JSON. RESTCONF relies on HTTP with TCP in contrast to CoMI which uses CoAP that is optimized for UDP with less overhead for small messages. RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not used by CoMI.

CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips.

To promote small packets, CoMI uses an additional "data-identifier string-to-number conversion" to minimize CBOR payloads and URI length.

Currently, small managed devices need to support at least two protocols: CoAP and SNMP [RFC3411]. When the MIB can be accessed with the CoMI protocol, the SNMP protocol can be replaced with the CoAP protocol. Although the SNMP server size is not huge (see Appendix A), the code for the security aspects of SMIV3 [RFC3414] is not negligible. Using CoAP to access secured management objects reduces the code complexity of the stack in the constrained device, and harmonizes applications development.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Readers of this specification should be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

The following terms are defined in the NETCONF protocol [RFC6241]: client, configuration data, data-store, and server.

The following terms are defined in the YANG data modelling language [RFC6020]: container, data node, key, key leaf, leaf, leaf-list, and list. The following terms are defined in RESTCONF protocol [I-D.ietf-netconf-restconf]: data resource, data-store resource, edit operation, query parameter, target resource, and unified data-store. The terms YANG hash, and Rehash bit are defined in I-D.yang-hash.

The following terms are defined in this document:

Data-node instance: An instance of a data-node specified in a YANG module present in the server. The instance is stored in the memory of the server.

Notification-node instance: An instance of a schema node of type notification, specified in a YANG module present in the server. The instance is generated in the server at the occurrence of the corresponding event and appended to a stream.

The following list contains the abbreviations used in this document.

XXXX: TODO, and others to follow.

1.1.1. Tree Diagrams

A simplified graphical representation of the data model is used in the YANG modules specified in this document. The meaning of the symbols in these diagrams is as follows:

Brackets "[" and "]" enclose list keys.

Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for the reading and modifying of instrumentation variables used for the management of the instrumented node.

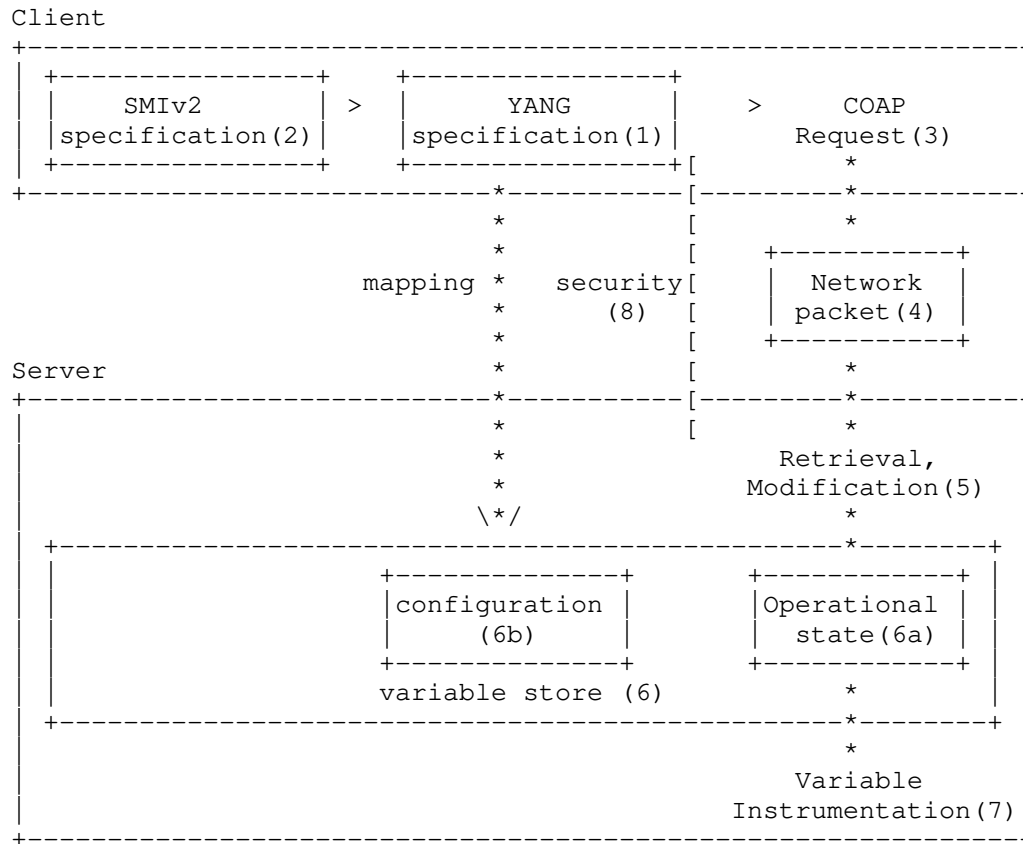


Figure 1: Abstract CoMI architecture

Figure 1 is a high level representation of the main elements of the CoAP management architecture. A client sends requests as payload in packets over the network to a managed constrained node.

Objectives are:

- o Equip a constrained node with a management server that provides information about the operational characteristics of the code running in the constrained node.
- o The server provides this information in a variable store that contains values describing the performance characteristics and the code parameter values.

- o The client receives the performance characteristics on a regular basis or on request.
- o The client sets the parameter values in the server at bootstrap and intermittently when operational conditions change.
- o The constrained network requires the payload to be as small as possible, and the constrained server memory requirements should be as small as possible.

For interoperability it is required that in addition to using the Internet Protocol for data transport:

- o The names, type, and semantics of the instrumentation variables are standardized.
- o The instrumentation variables are described in a standard language.
- o The URI of the CoAP request is standardized.
- o The format of the packet payload is standardized.
- o The notification from server to client is standardized.

The different numbered components of Figure 1 are discussed according to component number.

- (1) YANG specification: contains a set of named and versioned modules. A module specifies a hierarchy of named and typed resources. A resource is uniquely identified by a sequence of its name and the names of the enveloping resources following the hierarchy order. The YANG specification serves as input to the writers of application and instrumentation code and the humans analyzing the returned values (arrow from YANG specification to Variable store). The specification can be used to check the correctness of the CoAP request and do the CBOR encoding.
- (2) SMIV2 specification: A named module specifies a set of variables and "conceptual tables". Named variables have simple types. Conceptual tables are composed of typed named columns. The variable name and module name identify the variable uniquely. There is an algorithm to translate SMIV2 specifications to YANG specifications.
- (3) CoAP request: The CoAP request needs a Universal Resource Identifier (URI) and the payload of the packet to send a request. The URI is composed of the schema, server, path and query and

looks like `coap://entry.example.com/<path>?<query>`. Fragments are not supported. Allowed operations are PUT, PATCH, GET, DELETE, and POST. New variables can be created with POST when they exist in the YANG specification. The Observe option is used to return variable values regularly or on event occurrence (notification).

- (3.1) CoAP <path>: The path identifies the variable in the form `"/mg/<encoded-name>"`.
- (3.2) CoAP <query>: The query parameter is used to specify additional (optional) aspects like the container name, list instance, and others. The idea is to keep the path simple and put variations on variable specification in the query.
- (3.3) CoAP discovery: Discovery of the variables is done with standard CoAP resource discovery using `/.well-known/core` with `?rt=/core.mg`.
- (4) Network packet: The payload contains the CBOR encoding of JSON objects. This object corresponds with the converted RESTCONF message payload.
- (5) Retrieval, modification: The server needs to parse the CBOR encoded message and identify the corresponding instances in the Variable store. In addition, this component includes the code for CoAP Observe and block options.
- (6) Variable store: The store is composed of two parts: Operational state and Configuration data-store (see Section 2.1). CoMI does not differentiate between variable store types. The Variable store contains data-node instances. Values are stored in the appropriate instances, and or values are returned from the instances into the payload of the packet.
- (7) Variable instrumentation: This code depends on implementation of drivers and other node specific aspects. The Variable instrumentation code stores the values of the parameters into the appropriate places in the operational code. The variable instrumentation code reads current execution values from the operational code and stores them in the appropriate instances.
- (8) Security: The server MUST prevent unauthorized users from reading or writing any data resources. CoMI relies on DTLS [RFC6347] which is specified to secure CoAP communication.

2.1. RESTCONF/YANG Architecture

CoMI adapts the RESTCONF architecture so data exchange and implementation requirements are optimized for constrained devices.

The RESTCONF protocol uses a unified data-store to edit conceptual data structures supported by the server. The details of transaction preparation and non-volatile storage of the data are hidden from the RESTCONF client. CoMI also uses a unified data-store, to allow stateless editing of configuration variables and the notification of operational variables.

The child schema nodes of the unified data-store include all the top-level YANG data nodes in all the YANG modules supported by the server. The YANG data structures represent a hierarchy of data resources. The client discovers the list of YANG modules, and important conformance information such as the module revision dates, YANG features supported, and YANG deviations required. The individual data nodes are discovered indirectly by parsing the YANG modules supported by the server.

The YANG data definition statements contain a lot of information that can help automation tools, developers, and operators use the data model correctly and efficiently. The YANG definitions and server YANG module capability advertisements provide an "API contract" that allow a client to determine the detailed server management capabilities very quickly

RESTCONF and CoMI use a simple algorithmic mapping from YANG to URI syntax to identify the target resource of a retrieval or edit operation. A client can construct operations or scripts using a predictable syntax, based on the YANG data definitions. The target resource URI can reference a data resource instance, or the data-store itself (to retrieve the entire data-store or create a top-level data resource instance). A compression algorithm reduces the size of the data-node instance identifier (see Section 2.2).

2.1.1. Major differences between RESTCONF and CoMI

CoMI uses CoAP/UDP as transport protocol and CBOR as payload format. RESTCONF uses HTTP/TCP as transport protocol and JSON or XML as payload formats. CoMI encodes YANG name strings as numbers, where RESTCONF does not.

CoAP servers MUST maintain the order of user-ordered data. CoMI does not support insert-mode (first, last, before, after) and insertion-point (before, after) which are supported by RESTCONF. Many CoAP

servers will not support date and time functions. For that reason CoMI does not support the start, stop options for events.

The CoMI "select" query parameter is equivalent to the RESTCONF "fields" query parameter but has a much simpler syntax. CoMI servers only implement the efficient "trim" mode for default values. CoMI servers implement a less rich syntax to specify key values in the URI than RESTCONF servers.

CoMI servers do not support 'filter' query that involves XML parsing, 'content', 'depth', and 'with-defaults' query parameters.

CoMI servers do not support the YANG functionality of anyxml, anydata, and xpath.

2.2. Compression of data-node instance identifier

The JSON/CBOR encoding will include the module name string to specify the YANG module. If a representation of the target resource is included in the request or response message, then the data definition name string is used to identify each node in the message. The module namespace (or name) may also be present in these identifiers.

In order to significantly reduce the size of identifiers used in CoMI, numeric object identifiers are used instead of these strings. The specific encoding of the object identifiers is not hard-wired in the protocol.

YANG Hash is the default encoding for object identifiers [I-D.bierman-core-yang-hash]. This encoding is considered to be "unstructured" since the particular values for each object are determined by a hash algorithm. It is possible for 2 different objects to generate the same hash value. If this occurs, then the client and server will both need to rehash the colliding object identifiers to new unused hash values.

In order to eliminate the need for rehashing, CoMI allows for alternate "structured" object identifier encoding formats. Structured object identifier MUST be managed such that no object ID collisions are possible, and therefore no rehash procedures are needed. Structured object identifiers can also be selected to minimize the size of a subset of the object identifiers (e.g., the most requested objects).

3. CoAP Interface

In CoAP a group of links can constitute a Function Set. The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Function Set. CoMI end-points that implement the CoMI management protocol support at least one discoverable management resource of resource type (rt): core.mg, with path: /mg, where mg is short-hand for management. The name /mg is recommended but not compulsory (see Section 4.6).

The path prefix /mg has resources accessible with the following five paths:

/mg: YANG-based data with path "/mg" and using CBOR content encoding format. This path represents a data-store resource which contains YANG data resources as its descendant nodes. All identifiers referring to YANG data nodes within this path are encoded YANG names (see for example [I-D.bierman-core-yang-hash]).

/mg/mod.uri: URI identifying the location of the server module information, with path "/mg/mod.uri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG encoded values. An Entity Tag MUST be maintained for this resource by the server, which MUST be changed to a new value when the set of YANG modules in use by the server changes.

/mg/num.typ: String identifying the object ID numbering scheme used by the CoMI server. Two values are defined in this document: (1) 'yanghash' to indicate that the YANG Hash numbering scheme is used, and (2) 'yangmanag' to indicate that a managed numbering scheme is used. It is possible for other object numbering schemes to be defined outside the scope of this document.

/mg/srv.typ: String identifying the CoMI server type. The value 'ro' indicates that the server is a read-only server and no editing operations are supported. A read-only server is not required to provide YANG deviation statements for any writable YANG data nodes. The value 'rw' indicates that the server is a read-write server and editing operations are supported. A read-write server is required to provide YANG deviation statements for any writable YANG data nodes that are not fully implemented.

/mg/yh.uri: URI indicating the location of the server YANG hash information if any objects needed to be re-hashed by the server. It has the path "/mg/yh.uri" and is encoded in CBOR format. The "yang-hash" container within the "ietf-yang-hash" module, described in [I-D.bierman-core-yang-hash], is used to define the syntax and semantics of this data structure. This YANG data is

encoded with plain identifier strings, not YANG hash values. The server will only have this resource if there are any objects that needed to be re-hashed due to a hash collision.

/mg/stream: String identifying the default stream resource to which YANG notification instances are appended. Notification support is optional, so this resource will not exist if the server does not support any notifications.

/mg/op: String identifying the resource to which YANG operations are appended.

The mapping of YANG data node instances to CoMI resources is as follows: A YANG module describes a set of data trees composed of YANG data nodes. Every root of a data tree in a YANG module loaded in the CoMI server represents a resource of the server. All data root descendants represent sub-resources.

The resource identifiers of the instances of the YANG specifications are encoded YANG names. When multiple instances of a list node exist, the instance selection is described in Section 4.1.4.4

The profile of the management function set, with IF=core.mg, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

name	path	rt	Data Type
Management	/mg	core.mg	n/a
Data	/mg	core.mg.data	application/cbor
Module Set URI	/mg/mod.uri	core.mg.moduri	application/cbor
Numbering Type	/mg/num.typ	core.mg.num-type	application/cbor
Server Type	/mg/srv.typ	core.mg.srv-type	application/cbor
YANG Hash Info	/mg/yh.uri	core.mg.yang-hash	application/cbor
Events	/mg/stream	core.mg.stream	application/cbor
Operations	/mg/op	core.mg.op	application/cbor

4. MG Function Set

The MG Function Set provides a CoAP interface to perform a subset of the functions provided by RESTCONF.

A subset of the operations defined in RESTCONF are used in CoMI:

Operation	Description
GET	Retrieve the data-store resource or a data resource
POST	Create a data resource
PUT	Create or replace a data resource
PATCH	Replace a data resource partially
DELETE	Delete a data resource

4.1. Data Retrieval

4.1.1. GET

One or more instances of data resources are retrieved by the client with the GET method. The RESTCONF GET operation is supported in CoMI. The same constraints apply as defined in section 3.3 of [I-D.ietf-netconf-restconf]. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [I-D.ietf-core-block] is used, as explained in more detail in Section 4.5.

There are three query parameters for the GET method. A CoMI server MUST implement the keys parameter and the content parameter, and MAY implement the select parameter to allow common data retrieval filtering functionality.

Query Parameter	Description
keys	Request to select instances of a YANG definition
select	Request to select sub-trees from the target resource
content	Request to select configuration and non-configuration nodes

The "keys" parameter is used to specify a specific instance of the list resource. When keys is not specified, all instances are returned. When no or one instance of the resource exists, the keys parameter is ignored.

4.1.2. Using the 'select' Parameter

RESTCONF uses the 'filter' parameter next to the 'fields' parameter to specify an expression which can represent a subset of all data nodes within the target resource [I-D.ietf-netconf-restconf]. The 'select' parameter is local to CoMI and is useful for filtering sub-trees and retrieving only a subset that a managing application is interested in.

Because filtering is a resource intensive task and not all constrained devices can be expected to have enough computing resources such that they will be able to successfully filter and return a subset of a sub-tree. This is especially likely to be true with Class 0 devices that have significantly lesser RAM than 10 KiB [RFC7228]. Since CoMI is targeted at constrained devices and networks, the 'filter' parameter is not used here.

The implementation of the 'select' parameter is already optional for constrained devices, however, even when implemented it is expected to be a best effort feature, rather than a service that nodes must provide. This implies that if a node receives the 'select' parameter specifying a set of sub-trees that should be returned, it will only return those that it is able to return.

4.1.3. Using the 'content' query parameter

The "content" parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
config	Return only configuration descendant data nodes
nonconfig	Return only non-configuration descendant data nodes
all	Return all descendant data nodes

This parameter is only allowed for GET methods on datastore and data resources. A 4.00 Bad Request error is returned if used for other methods or resource types.

The default value is determined by the "config" statement value of the requested data nodes. If the "config" value is "false", then the default for the "content" parameter is "nonconfig". If "config" is "true" then the default for the "content" parameter is "config".

4.1.4. Retrieval Examples

In all examples the path is expressed in readable names and as a encoded value of the name (where the encoded value in the payload is expressed as a hexadecimal number, and the encoded value in the URL as a base64 number). CoMI payloads use the CBOR format. The CBOR syntax of the YANG payloads is specified in TODO REF. The examples in this section use a JSON payload with extensions to approach the permissible CBOR payload, called "diagnostic JSON".

4.1.4.1. Single instance retrieval

A request to read the values of instances of a management object or the leaf of an object is sent with a confirmable CoAP GET message. A single object is specified in the URI path prefixed with /mg.

Using for example the clock container from [RFC7317], a request is sent to retrieve the value of clock/current-datetime specified in module system-state. The answer to the request returns a (identifier, value) pair, transported as a CBOR map with a single item.

REQ: GET example.com/mg/system-state/clock/current-datetime

RES: 2.05 Content (Content-Format: application/cbor)

```
{
  "current-datetime" : "2014-10-26T12:16:31Z"
}
```

The specified object can be an entire object. Accordingly, the returned payload is composed of all the leaves associated with the object. The payload is a CBOR map where each leaf is returned as a (encoded YANG name, value) pair. For example, the GET of the clock object, sent by the client, results in the following returned payload sent by the managed entity, transported as A CBOR map with two items:

```
REQ: GET example.com/mg/system-state/clock
      (Content-Format: application/cbor)
```

```
RES: 2.05 Content (Content-Format: application/cbor)
{
  "clock" : {
    "current-datetime" : "2014-10-26T12:16:51Z",
    "boot-datetime" : "2014-10-21T03:00:00Z"
  }
}
```

For example, the YANG names can be replaced by the hash values for 'clock', 'current-datetime', and 'boot-datetime'. The 30 bit murmur3 hash value of 'clock' equates: CDKSQ (xref target="I-D.bierman-core-yang-hash"/>. The request using hash values is shown below:

```
REQ: GET example.com/mg/CDKSQ
      (Content-Format: application/cbor)
```

```
RES: 2.05 Content (Content-Format: application/cbor)
{
  0x021ca491 : {
    0x047c468b : "2014-10-26T12:16:51Z",
    0x1fb5f4f8 : "2014-10-21T03:00:00Z"
  }
}
```

4.1.4.2. Multiple instance retrieval

A "list" node can have multiple instances. Accordingly, the returned payload is composed of all the instances associated with the list node. Each instance is returned as a (key, object) pair, where key and object are composed of one or more (identifier, value) pairs.

For example, the GET of the /interfaces/interface/ipv6/neighbor results in the following returned payload sent by the managed entity, transported as a CBOR map of 3 (key : object) pairs, where key and

value are CBOR maps with one entry each. In this case the key is the "ip" attribute and the value is the "link-layer-address" attribute.

REQ: GET example.com/mg/interfaces/interface/ipv6/neighbor
(Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)

```
{
  "neighbor" :{
    {"ip" : "fe80::200:f8ff:fe21:67cf"}:
      {"link-layer-address" : "00:00::10:01:23:45"}
    ,
    {"ip" : "fe80::200:f8ff:fe21:6708"}:
      {"link-layer-address" : "00:00::10:54:32:10"}
    ,
    {"ip" : "fe80::200:f8ff:fe21:88ee"}:
      {"link-layer-address" : "00:00::10:98:76:54"}
  }
}
```

4.1.4.3. Access to MIB Data

The YANG translation of the SMI specifying the ipNetToMediaTable [RFC4293] yields:


```
container IP-MIB {
  container ipNetToPhysicalTable {
    list ipNetToPhysicalEntry {
      key "ipNetToPhysicalIfIndex
          ipNetToPhysicalNetAddressType
          ipNetToPhysicalNetAddress";
      leaf ipNetToMediaIfIndex {
        type: int32;
      }
      leaf ipNetToPhysicalIfIndex {
        type if-mib:InterfaceIndex;
      }
      leaf ipNetToPhysicalNetAddressType {
        type inet-address:InetAddressType;
      }
      leaf ipNetToPhysicalNetAddress {
        type inet-address:InetAddress;
      }
      leaf ipNetToPhysicalPhysAddress {
        type yang:phys-address {
          length "0..65535";
        }
      }
      leaf ipNetToPhysicalLastUpdated {
        type yang:timestamp;
      }
      leaf ipNetToPhysicalType {
        type enumeration { ... }
      }
      leaf ipNetToPhysicalState {
        type enumeration { ... }
      }
      leaf ipNetToPhysicalRowStatus {
        type snmpv2-tc:RowStatus;
      }
    }
  }
}
```

The following example shows an "ipNetToPhysicalTable" with 2 instances, using JSON encoding as defined in [I-D.ietf-netmod-yang-json]:

```

{
  "IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry" : [
    {
      "ipNetToPhysicalIfIndex" : 1,
      "ipNetToPhysicalNetAddressType" : "ipv4",
      "ipNetToPhysicalNetAddress" : "10.0.0.51",
      "ipNetToPhysicalPhysAddress" : "00:00:10:01:23:45",
      "ipNetToPhysicalLastUpdated" : "2333943",
      "ipNetToPhysicalType" : "static",
      "ipNetToPhysicalState" : "reachable",
      "ipNetToPhysicalRowStatus" : "active"
    },
    {
      "ipNetToPhysicalIfIndex" : 1,
      "ipNetToPhysicalNetAddressType" : "ipv4",
      "ipNetToPhysicalNetAddress" : "9.2.3.4",
      "ipNetToPhysicalPhysAddress" : "00:00:10:54:32:10",
      "ipNetToPhysicalLastUpdated" : "2329836",
      "ipNetToPhysicalType" : "dynamic",
      "ipNetToPhysicalState" : "unknown",
      "ipNetToPhysicalRowStatus" : "active"
    }
  ]
}

```

4.1.4.4. The 'keys' Query Parameter

There is a query parameter that MUST be supported by servers called "keys". This parameter is used to specify the key values for an instance of an object identified by an encoded YANG name. All key leaf values of the instance are passed in order. The first key leaf in the top-most list is the first key encoded in the 'keys' parameter.

The key leafs from top to bottom and left to right are encoded as a comma-delimited list. If a key leaf value is missing then all values for that key leaf are returned.

Example: In this example exactly one instance is requested from the ipNetToPhysicalEntry (from a previous example). The CBOR payload, here represented with diagnostic JSON, permits to transport the selected instance and nothing more.

TODO refer to the section in YANG to CBOR mapping

REQ: GET example.com/mg/IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry?keys=1,ipv4,9.2.3.4

RES: 2.05 Content (Content-Format: application/cbor)

```
{
  "IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry": {
    {
      "ipNetToPhysicalIfIndex" : 1,
      "ipNetToPhysicalNetAddressType" : "ipv4",
      "ipNetToPhysicalNetAddress" : "9.2.3.4":
    {
      "ipNetToPhysicalPhysAddress" : "00:00:10:54:32:10",
      "ipNetToPhysicalLastUpdated" : "2329836",
      "ipNetToPhysicalType" : "dynamic",
      "ipNetToPhysicalState" : "unknown",
      "ipNetToPhysicalRowStatus" : "active"
    }
  }
}
```

An example illustrates the syntax of keys query parameter. In this example the following YANG module is used:

```
module foo-mod {
  namespace foo-mod-ns;
  prefix foo;

  list A {
    key "key1 key2";
    leaf key1 { type string; }
    leaf key2 { type int32; }
    list B {
      key "key3";
      leaf key3 { type string; }
      leaf coll { type uint32; }
    }
  }
}
```

The following string represents the CoMI target resource identifier for the instance of the "coll" leaf with key values "top", 17, "group":

```
/mg/foo-mod:A/B/coll?keys="top",17,"group1"
```

4.1.4.5. The 'select' Query Parameter

The select parameter is used along with the GET method to provide a sub-tree filter mechanism. A list of encoded YANG names that should be filtered is provided along with a list of keys identifying the instances that should be returned. When the keys parameter is used together with the select, the key values are added in brackets without using the "keys=" text.

Data may be retrieved using the select query parameter in the following way, transported as a CBOR maps of maps:

REQ: GET example.com/mg/IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry?select=(10.0.0.51)

RES: 2.05 Content (Content-Format: application/cbor)

```
{
  "IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry": {
    {
      "ipNetToPhysicalIfIndex" : 1,
      "ipNetToPhysicalNetAddressType" : "ipv4",
      "ipNetToPhysicalNetAddress" : "10.0.0.51"}:
    {
      "ipNetToPhysicalPhysAddress" : "00:00:10:01:23:45",
      "ipNetToPhysicalLastUpdated" : "2333943",
      "ipNetToPhysicalType" : "static",
      "ipNetToPhysicalState" : "reachable",
      "ipNetToPhysicalRowStatus" : "active"
    }
  }
}
```

In this example exactly one instance is returned as response from the ipNetToPhysicalTable because only this instance matches the provided keys.

Supposing there were multiple YANG fields with their own sets of keys that were to be filtered, the select query parameter can be used to retrieve results from these in one go as well. Using the "foo-mod" module of Section 4.1.4.4, the following string represents the CoMI target resource identifier when multiple fields, with their own sets of key values are queried:

```
/mg/foo-mod:A?select=B/coll("top",17,"group"),key2("top")
```

4.1.4.6. Defaults handling

If the target of a GET method is a data node that represents a leaf that has a default value, and the leaf has not been given a value yet, the server MUST not return the leaf.

If the target of a GET method is a data node that represents a container or list that has any child resources with default values, for the child resources that have not been given value yet, the server MUST not return the child resource.

4.2. Data Editing

CoMI allows data-store contents to be created, modified and deleted using CoAP methods.

Data-editing is an optional feature. The server will indicate its editing capability with the `"/core.mg.srv-type` resource type. If the value is `'rw'` then the server supports editing operations. If the value is `'ro'` then the server does not support editing operations.

4.2.1. Data Ordering

A CoMI server is not required to support entry insertion of lists and leaf-lists that are ordered by the user (i.e., YANG statement `"ordered-by user"`). The `'insert'` and `'point'` query parameters from RESTCONF are not used in CoMI.

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as arrays so messages will preserve their order.

4.2.2. POST

Data resource instances are created with the POST method. The RESTCONF POST operation is supported in CoMI for creation of data resources and the invocation operation resources. Refer to Section 4.4 for details on operation resources. The same constraints apply as defined in section 4.4.1 of [I-D.ietf-netconf-restconf]. The operation is mapped to the POST method defined in section 5.8.2 of [RFC7252].

There are no query parameters for the POST method.

4.2.3. PUT

Data resource instances are created or replaced with the PUT method. The PUT operation is supported in CoMI. A request to set the values of instances of an object/leaf is sent with a confirmable CoAP PUT message. The Response is piggybacked to the CoAP ACK message corresponding with the Request. The same constraints apply as defined in section 3.5 of [I-D.ietf-netconf-restconf]. The operation is mapped to the PUT method defined in section 5.8.3 of [RFC7252].

There are no query parameters for the PUT method.

4.2.4. PATCH

Data resource instances are partially replaced with the PATCH method [I-D.vanderstok-core-patch]. The PATCH operation is supported in CoMI. A request to set the values of instances of a subset of the values of the resource is sent with a confirmable CoAP PATCH message. The Response is piggybacked to the CoAP ACK message corresponding with the Request. The same constraints apply as defined in section 3.5 of [I-D.ietf-netconf-restconf]. The operation is mapped to the PATCH method defined in [I-D.vanderstok-core-patch].

The processing of the PATCH command is specified by the CBOR payload. The CBOR patch payload describes the changes to be made to target YANG data nodes. It follows closely the rules described in [RFC7396]. If the CBOR patch payload contains objects that are not present in the target, these objects are added. If the target contains the specified object, the contents of the objects are replaced with the values of the payload. Null values indicate the removal of existing values. The CBOR patch extends [RFC7396] by specifying rules for list elements.

TODO, review text after publication of YANG/CBOR and CBOR-merge drafts.

For example consider the following YANG specification:

```
module foo {  
  namespace "http://example.com/book";  
  prefix "bo";  
  revision 2015-06-07;  
  
  list B {  
    key key1;  
    key key2;  
    leaf key1 { type string; }  
    leaf key2 {type string; }  
    leaf coll { type int32; }  
    leaf counter1 { type uint32; }  
  }  
  
  container book {  
    leaf title { type string; }  
    container author {  
      leaf givenName {type string; }  
      leaf familyName {type string; }  
    }  
    leaf-list tags {type string; }  
    leaf content{type string;}  
    leaf phoneNumber {type string;}  
  }  
}
```

Consider the following target data nodes described with the JSON encoding of [I-D.ietf-netmod-yang-json].

```
"B": [
  {
    "key1" : "author1",
    "key2" : "book2",
    "col1" : 25,
    "counter1" : 4321
  },
  {
    "key1" : "author5",
    "key2" : "book6",
    "col1" : 2,
    "counter1" : 1234
  }
]

"book": {
  "title" : "mytitle",
  "author": {
    "givenName" : "John",
    "familyName" : "Doe"
  }
  "tags" : [ "example", "sample"],
  "content" : "This will be unchanged"
}
```

The following changes are requested for the document (following the example from [RFC7396]: the title changes from "mytitle" to "favoured", the phoneNumber is added to the book container, the familyName is deleted, and "sample" is removed from the tags leaf-list. In addition author1, book1 item is removed, author5 counter1 is upgraded, and a new author is added in B list. The following CBOR Patch payload, represented in JSON is sent.

TODO: edit after publication of CBOR-merge draft.


```
{
  "B": {
    { "key1" : "author1",
      "key2" : "book2"}:
    { null : null},
    { "key1" : "author5"} :
      {"counter1" : 4444},
    { "key1" : "newauthor",
      "key2" : "newbook"}:
    { "coll" : 1,
      "counter1" : 1}
  },
  "book" : {
    "title" : "favoured",
    "author": {"familyName" : null},
    "tags" : [ "example"],
    "phoneNumber" : "+01-123-456-7890"
  }
}
```

In his example, the value "author5" specifies the entry uniquely. However, when several entries exist with the "author5" value for "key1", the outcome of the example Patch is undefined.

The processing of the Patch payload results in the following new target data nodes.

```
"B": [
  {
    "key1" : "newauthor",
    "key2" : "newbook",
    "col1" : 1,
    "counter1" : 1
  },
  {
    "key1" : "author5",
    "key2" : "book6",
    "col1" : 2,
    "counter1" : 4444
  }
]

"book": {
  "title" : "favoured",
  "author": {
    "givenName" : "John"
  }
  "tags" : [ "example"],
  "content" : "This will be unchanged",
  "phoneNumber" : "+01-123-456-7890"
}
```

There are no query parameters for the PATCH method.

4.2.5. DELETE

Data resource instances are deleted with the DELETE method. The RESTCONF DELETE operation is supported in CoMI. The same constraints apply as defined in section 3.7 of [I-D.ietf-netconf-restconf]. The operation is mapped to the DELETE method defined in section 5.8.4 of [RFC7252].

There are no optional query parameters for the DELETE method.

4.2.6. Editing Multiple Resources

Editing multiple data resources at once can allow a client to use fewer messages to make a configuration change. It also allows multiple edits to all be applied or none applied, which is not possible if the data resources are edited one at a time.

It is easy to add multiple entries at once. The "PATCH" method can be used to simply patch the parent node(s) of the data resources to be added. If multiple top-level data resources need to be added, then the data-store itself ('/mg') can be patched.

If other operations need to be performed, or multiple operations need to be performed at once, then the YANG Patch

[I-D.ietf-netconf-yang-patch] media type can be used with the PATCH method. A YANG patch is an ordered list of edits on the target resource, which can be a specific data node instance, or the data-store itself. The resource type used by YANG Patch is 'application/yang.patch'. A status message is returned in the response, using resource type 'application/yang.patch.status'.

The following YANG tree diagram describes the YANG Patch structure, Each 'edit' list entry has its own operation, sub-resource target, and new value (if needed).

```
+--rw yang-patch
  +--rw patch-id?   string
  +--rw comment?    string
  +--rw edit* [edit-id]
    +--rw edit-id      string
    +--rw operation    enumeration
    +--rw target       target-resource-offset
    +--rw point?       target-resource-offset
    +--rw where?       enumeration
    +--rw value
```

Refer to [I-D.ietf-netconf-yang-patch] for more details on the YANG Patch request and response contents.

4.3. Notify functions

Notification by the server to a selection of clients when an event occurs in the server is an essential function for the management of servers. CoMI allows events specified in YANG [RFC5277] to be notified to a selection of requesting clients. The server appends newly generated events to a stream. There is one, so-called "default", stream in a CoMI server. The /mg/stream resource identifies the default stream. The server MAY create additional streams. When a CoMI server generates an internal event, it is appended to the chosen stream, and the contents of a notification instance is ready to be sent to all CoMI clients which observe the chosen stream resource.

Reception of generated notification instances is enabled with the CoAP Observe [I-D.ietf-core-observe] function. The client subscribes to the notifications by sending a GET request with an "Observe" option, specifying the /mg/stream resource when the default stream is selected.

Every time an event is generated, the chosen stream is cleared, and the generated notification instance is appended to the chosen stream. After appending the instance, the contents of the instance is sent to all clients observing the modified stream.

Suppose the server generates the event specified with:

```
module example-port {
  ...
  prefix ep;
  ...
  notification example-port-fault {
    description
      "Event generated if a hardware fault on a
       line card port is detected";
    leaf port-name {
      type string;
      description "Port name";
    }
    leaf port-fault {
      type string;
      description "Error condition detected";
    }
  }
}
```

By executing a GET on the /mg/stream resource the client receives the following response:

```
REQ: GET example.com/mg/stream
      (observe option register)

RES: 2.05 Content (Content-Format: application/cbor)
{
  "example-port-fault":{
    "port-name" : "0/4/21",
    "port-fault" : "Open pin 2"
  }
}
```

In the example, the request returns a success response with the contents of the last generated event. Consecutively the server will regularly notify the client when a new event is generated.

To check that the client is still alive, the server MUST send confirmable notifications once in a while. When the client does not confirm the notification from the server, the server will remove the client from the list of observers [I-D.ietf-core-observe].

In the registration request, the client MAY include a "Response-To-Uri-Host" and optionally "Response-To-Uri-Port" option as defined in [I-D.becker-core-coap-sms-gprs]. In this case, the observations SHOULD be sent to the address and port indicated in these options. This can be useful when the client wants the managed device to send the trap information to a multicast address.

4.4. RPC statements

An operation resource represents a protocol operation defined with the YANG "rpc" statement. It is invoked using a POST method on the operation resource with a Token value as specified in section 5.3 "Request/Response matching" of [RFC7252] to match the operation request sent by the RPC requester to the Operation request sent by the RPC executor.

```
POST mg/op/<operation>
```

The <operation> field identifies the module name and rpc identifier string for the desired operation.

For example, if "module-A" defined a "reset" operation, then invoking the operation from "module-A" would be requested as follows:

```
POST example.com/mg/op/module-A:reset
```

If the "rpc" statement has input parameters, then a message-body MAY be sent by the client in the request, otherwise the request message MUST NOT include a message-body.

If the operation is successfully invoked the server MUST send a 2.04 Changed status code. If the operation is not successfully invoked, then a message-body SHOULD be sent by the server, containing an error, as defined in Section 4.7.

If the "rpc" statement has return parameters, then the server invokes a POST method with the same Token value as used in the request from the client. The payload contains the values of the return parameters.

4.4.1. Encoding RPC input parameters

If the "rpc" statement has an "input" section, then the "input" node is provided in the message-body, corresponding to the YANG data definition statements within Request payload.

The following YANG definition is used for the examples in this section.

```
module example-ops {
  namespace "https://example.com/ns/example-ops";
  prefix "ops";

  rpc reboot {
    input {
      leaf delay {
        units seconds;
        type uint32;
        default 0;
      }
      leaf message { type string; }
      leaf language { type string; }
    }
  }

  rpc get-reboot-info {
    output {
      leaf reboot-time {
        units seconds;
        type uint32;
      }
      leaf message { type string; }
      leaf language { type string; }
    }
  }
}
```

The client might send the following POST request message:

```
POST example.com/restconf/operations/example-ops:reboot
Token:0x56
Content-Type:
{ "example-ops:input":
  {
    "delay": 600,
    "message": "Going down for system maintenance"
    "language": "en-US"
  }
}
```

The server may respond with a 2.04 Changed.

4.4.2. Encoding RPC output parameters

If the "rpc" statement has output parameters, then the "output" node is provided in a PUT message, corresponding to the YANG data definition statements within the "output" section.

The "example-ops" YANG module defined Section 4.4 is used for the examples in this section.

The client might send the following POST request message:

```
POST example.com/mg/op/example-ops:get-reboot-info
Token: 0x56
```

The server might respond with a POST request that is related to the original RPC invocation by the Token value:

```
POST [ip:port]/mg/op/example-ops:output
Token: 0x56
{"example-ops:output" :
  {
    "reboot-time" : 30,
    "message" : "Going down for system maintenance",
    "language" : "en-US"
  }
}
```

4.5. Use of Block

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of text need to be transported the datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block

option [I-D.ietf-core-block] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying fragment sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

The block size is specified as exponents of the power 2. The SZX exponent value can have 7 values ranging from 0 to 6 with associated block sizes given by $2^{(SZX+4)}$; for example SZX=0 specifies block size 16, and SZX=3 specifies block size 128.

The block number of the block to transmit can be specified. There are two block options: Block1 option for the request payload transported with PUT, POST or PATCH, and the block2 option for the response payload with GET. Block1 and block2 can be combined. Examples showing the use of block option in conjunction with observer options are provided in [I-D.ietf-core-block].

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

Beware of race conditions. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. In the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the variables, the transported results do not correspond with a state having occurred in the server; or worse the returned values are inconsistent. For example: array length does not correspond with actual number of items. It may be advisable to use CBOR maps or CBOR arrays of undefined length which are foreseen for data streaming purposes.

4.6. Resource Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to `"/.well-known/core"` including a resource type (RT) parameter with the value `"core.mg"` [RFC6690]. Upon success, the return payload will contain the root resource of the management data. It is up to the implementation to choose its root resource, but it is recommended that the value `"/mg"` is used, where possible. The example below shows the discovery of the presence and location of management data.


```
REQ: GET /.well-known/core?rt=core.mg
```

```
RES: 2.05 Content </mg>; rt="core.mg"
```

Management objects MAY be discovered with the standard CoAP resource discovery. The implementation can add the encoded values of the object identifiers to `/.well-known/core` with `rt="core.mg.data"`. The available objects identified by the encoded values can be discovered by sending a GET request to `/.well-known/core` including a resource type (RT) parameter with the value `"core.mg.data"`. Upon success, the return payload will contain the registered encoded values and their location. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.mg.data
```

```
RES: 2.05 Content </mg/BaAiN>; rt="core.mg.data",  
</mg/CF_fA>; rt="core.mg.data"
```

Lists of encoded values may become prohibitively long. It is discouraged to provide long lists of objects on discovery. Therefore, it is recommended that details about management objects are discovered by reading the YANG module information stored in the `"ietf-yang-library"` module [I-D.ietf-netconf-restconf]. The resource `"/mg/mod.uri"` is used to retrieve the location of the YANG module library.

The module list can be stored locally on each server, or remotely on a different server. The latter is advised when the deployment of many servers are identical.

Local in example.com server:

REQ: GET example.com/mg/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
 "mod.uri" : "example.com/mg/modules"
}

Remote in example-remote-server:

REQ: GET example.com/mg/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
 "moduri" : "example-remote-server.com/mg/group17/modules"
}

Within the YANG module library all information about the module is stored such as: module identifier, identifier hierarchy, grouping, features and revision numbers.

The hash identifier is obtained as specified in [I-D.bierman-core-yang-hash]. When a collision occurred in the name space of the target server, a rehash is executed.

4.7. Error Return Codes

The RESTCONF return status codes defined in section 6 of the RESTCONF draft are used in CoMI error responses, except they are converted to CoAP error codes.

RESTCONF Status Line	CoAP Status Code
100 Continue	none?
200 OK	2.05
201 Created	2.01
202 Accepted	none?
204 No Content	2.04 Changed
304 Not Modified	2.03
400 Bad Request	4.00
403 Forbidden	4.03
404 Not Found	4.04
405 Method Not Allowed	4.05
409 Conflict	none?
412 Precondition Failed	4.12
413 Request Entity Too Large	4.13
414 Request-URI Too Large	4.00
415 Unsupported Media Type	4.15
500 Internal Server Error	5.00
501 Not Implemented	5.01
503 Service Unavailable	5.03

5. Error Handling

In case a request is received which cannot be processed properly, the managed entity **MUST** return an error message. This error message **MUST** contain a CoAP 4.xx or 5.xx response code, and **SHOULD** include additional information in the payload.

Such an error message payload is encoded in CBOR, using the following structure:

TODO: Adapt RESTCONF <errors> data structure for use in CoMI. Need to select the most important fields like <error-path>.

```
errorMsg      : ErrorMsg;
```

```
*ErrorMsg {
  errorCode   : uint;
  ?errorText  : tstr;
}
```

The variable "errorCode" has one of the values from the table below, and the OPTIONAL "errorText" field contains a human readable explanation of the error.

CoMI Error Code	CoAP Error Code	Description
0	4.00	General error
1	4.00	Malformed CBOR data
2	4.00	Incorrect CBOR datatype
3	4.00	Unknown MIB variable
4	4.00	Unknown conversion table
5	4.05	Attempt to write read-only variable
0..2	5.01	Access exceptions
0..18	5.00	SMI error status

The CoAP error code 5.01 is associated with the exceptions defined in [RFC3416] and CoAP error code 5.00 is associated with the error-status defined in [RFC3416].

6. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. This requires integrity protection of both requests and responses, and depending on the application encryption.

CoMI re-uses the security mechanisms already available to CoAP as much as possible. This includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorization mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorization may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However some adaptations may still be required, to cater for CoMI's specific requirements.

7. IANA Considerations

'rt="core.mg.data"' needs registration with IANA.

'rt="core.mg.moduri"' needs registration with IANA.

'rt="core.mg.num-type"' needs registration with IANA.

'rt="core.mg.srv-type"' needs registration with IANA.

'rt="core.mg.yang-hash"' needs registration with IANA.

'rt="core.mg.stream"' needs registration with IANA.

'rt="core.mg.op"' needs registration with IANA.

Content types to be registered:

- o application/comi+cbor

8. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification and specified CBOR encoding and use of hashes. Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR. The draft has benefited from comments (alphabetical order) by Somaraju Abhinav, Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Alexander Pelov, Juergen Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michel Veillette, Michael Verschoor, and Thomas Watteyne.

9. Changelog

Changes from version 00 to version 01

- o Focus on MIB only
- o Introduced CBOR, JSON, removed BER
- o defined mappings from SMI to xx
- o Introduced the concept of addressable table rows

Changes from version 01 to version 02

- o Focus on CBOR, used JSON for examples, removed XML and EXI
- o added uri-query attributes mod and con to specify modules and contexts
- o Definition of CBOR string conversion tables for data reduction
- o use of Block for multiple fragments
- o Error returns generalized
- o SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- o Added security considerations

Changes from version 03 to version 04

- o Added design considerations section
- o Extended comparison of management protocols in introduction

- o Added automatic generation of CBOR tables
- o Moved lowpan table to Appendix

Changes from version 04 to version 05

- o Merged SNMP access with RESTCONF access to management objects in small devices
- o Added CoMI architecture section
- o Added RESTCONF NETMOD description
- o Rewrote section 5 with YANG examples
- o Added server and payload size appendix
- o Removed Appendix C for now. It will be replaced with a YANG example.

Changes from version 04 to version 05

- o Extended examples with hash representation
- o Added keys query parameter text
- o Added select query parameter text
- o Better separation between specification and instance
- o Section on discovery updated
- o Text on rehashing introduced
- o Elaborated SMI MIB example
- o Yang library use described
- o use of BigEndian/LittleEndian in Hash generation specified

Changes from version 05 to version 06

- o Hash values in payload as hexadecimal and in URL in base64 numbers
- o Streamlined CoMI architecture text
- o Added select query parameter text

- o Data editing optional
- o Text on Notify added
- o Text on rehashing improved with example

Changes from version 06 to version 07

- o reduced payload size by removing JSON hierarchy
- o changed rehash handling to support small clients
- o added LWM2M comparison
- o Notification handling as specified in YANG
- o Added Patch function
- o Rehashing completely reviewed
- o Discover type of YANG name encoding
- o Added new resource types
- o Read-only servers introduced
- o Multiple updates explained

Changes from version 07 to version 08

- o Changed YANG Hash algorithm to use module name instead of prefix
- o Added rehash bit to allow return values to identify rehashed nodes in the response
- o Removed /mg/mod.set resource since this is not needed
- o Clarified that YANG Hash is done even for unimplemented objects
- o YANG lists transported as CBOR maps of maps
- o Adapted examples with more CBOR explanation
- o Added CBOR code examples in new appendix
- o Possibility to use other than default stream
- o Added text and examples for Patch payload

- o Repaired some examples
- o Added appendices on hash clash probability and hash clash storage overhead

Changes from version 08 to version 09

- o Removed hash and YANG to CBOR sections
- o removed hashes from examples.
- o Added RPC
- o Added content query parameter.
- o Added default handling.
- o Listed differences with RESTCONF

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<http://www.rfc-editor.org/info/rfc5277>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <<http://www.rfc-editor.org/info/rfc7396>>.
- [I-D.becker-core-coap-sms-gprs]
Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-05 (work in progress), August 2014.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-16 (work in progress), December 2014.
- [I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-08 (work in progress), February 2016.
- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-09 (work in progress), December 2015.
- [I-D.vanderstok-core-patch]
Stok, P. and A. Sehgal, "Patch Method for Constrained Application Protocol (CoAP)", draft-vanderstok-core-patch-02 (work in progress), October 2015.
- [I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-07 (work in progress), December 2015.
- [I-D.bierman-core-yang-hash]
Bierman, A. and P. Stok, "YANG Hash", draft-bierman-core-yang-hash-00 (work in progress), February 2016.

10.2. Informative References

- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <<http://www.rfc-editor.org/info/rfc2578>>.
- [RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <<http://www.rfc-editor.org/info/rfc3410>>.
- [RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, DOI 10.17487/RFC3411, December 2002, <<http://www.rfc-editor.org/info/rfc3411>>.
- [RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, DOI 10.17487/RFC3414, December 2002, <<http://www.rfc-editor.org/info/rfc3414>>.
- [RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002, <<http://www.rfc-editor.org/info/rfc3416>>.
- [RFC3418] Presuhn, R., Ed., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, DOI 10.17487/RFC3418, December 2002, <<http://www.rfc-editor.org/info/rfc3418>>.
- [RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <<http://www.rfc-editor.org/info/rfc4293>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.

- [RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <<http://www.rfc-editor.org/info/rfc6643>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.
- [I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., and M. Koster, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-04 (work in progress), October 2015.
- [I-D.ietf-lwig-coap]
Kovatsch, M., Bergmann, O., and C. Bormann, "CoAP Implementation Guidance", draft-ietf-lwig-coap-03 (work in progress), July 2015.
- [XML] "Extensible Markup Language (XML)",
Web <http://www.w3.org/xml>.
- [OMA] "OMA-TS-LightweightM2M-V1_0-20131210-C", Web
http://technical.openmobilealliance.org/Technical/current_releases.aspx.
- [DTLS-size]
Hummen, R., Shafagh, H., Raza, S., Voigt, T., and K. Wehrle, "Delegation-based Authentication and Authorization for the IP-based Internet of Things", Web
http://www.vs.inf.ethz.ch/publ/papers/mshafagh_secon14.pdf.
- [mibreg] "Structure of Management Information (SMI) Numbers (MIB Module Registrations)", Web
<http://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml/>.

- [management] Schoenwalder, J. and A. Sehgal, "Management of the Internet of Things", Web <http://cnds.eecs.jacobs-university.de/slides/2013-im-iot-management.pdf>, 2013.
- [dcaf] Bormann, C., Bergmann, O., and S. Gerdes, "Delegated Authenticated Authorization for Constrained Environments", Private Information .
- [openwsn] Watteijne, T., "Coap size in Openwsn", Web <http://builder.openwsn.org/>.
- [Erbium] Kovatsch, M., "Erbium Memory footprint for coap-18", Private Communication .

Appendix A. Payload and Server sizes

This section provides information on code sizes and payload sizes for a set of management servers. Approximate code sizes are:

Code	processor	Text	Data	reference
Observe agent	erbium	800	n/a	[Erbium]
CoAP server	MSP430	1K	6	[openwsn]
SNMP server	ATmega128	9K	700	[management]
Secure SNMP	ATmega128	30K	1.5K	[management]
DTLS server	ATmega128	37K	2K	[management]
NETCONF	ATmega128	23K	627	[management]
JSON parser	CC2538	4.6K	8	[dcaf]
CBOR parser	CC2538	1.5K	2.6K	[dcaf]
DTLS server	ARM7	15K	4	[I-D.ietf-lwig-coap]
DTLS server	MSP430	15K	4	[DTLS-size]
Certificate	MSP430	23K		[DTLS-size]
Crypto	MSP430	2-8K		[DTLS-size]

Thomas says that the size of the CoAP server is rather arbitrary, as its size depends mostly on the implementation of the underlying library modules and interfaces.

Payload sizes are compared for the following request payloads, where each attribute value is null (N.B. these sizes are educated guesses, will be replaced with generated data). The identifier are assumed to be a string representation of the OID. Sizes for SysUpTime differ due to preambles of payload. "CBOR opt" stands for CBOR payload where the strings are replaced by table numbers.

Request	BERR SNMP	JSON	CBOR	CBOR opt
IPnetTOMediaTable	205	327	~327	~51
lowpanIfStatsTable		710	614	121
sysUpTime	29	13	~13	20
RESTCONF example				

Appendix B. Comparison with LWM2M

CoMI and LWM2M, both, provide RESTful device management services over CoAP. Differences between the designs are highlighted in this section.

LWM2M [OMA] objects are defined by standardized numbers. When new types are needed, new numbers need to be defined. This is the major difference with CoMI and YANG, where new modules can incorporate any type that is required without going through a standardization process, but may lead to rehashing. On the one hand LWM2M is static with very small numbered objects, where CoMI with YANG is more dynamic, with a number conversion overhead.

Unlike CoMI, which enables the use of SMIV2 and YANG data models for device management, LWM2M defines a new object resource model. This means that data models need to be redefined in order to use LWM2M. In contrast, CoMI provides access to a large variety of SMIV2 and YANG data modules that can be used immediately.

Objects and resources within CoMI are identified with a YANG hash value, however, each object is described as a link in the CoRE Link Format by LWM2M. This approach by LWM2M can lead to larger complex URIs and more importantly payloads can grow large in size. Using a hash value to represent the objects and resources allows URIs and

payloads to be smaller in size, which is important for constrained devices that may not have enough resources to process large messages.

LWM2M encodes payload data in Type-length-value (TLV), JSON or plain text formats. While the TLV encoding is binary and can result in reduced message sizes, JSON and plain text are likely to result in large message sizes when lots of resources are being monitored or configured. Furthermore, CoMI's use of CBOR gives it an advantage over the LWM2M's TLV encoding as well since this too is more efficient [citation needed].

CoMI is aligned with RESTCONF for constrained devices and uses YANG data models that have objects containing resources organized in a tree-like structure. On the other hand, LWM2M uses a very flat data model that follows the "object/instance/resource" format, with no possibility to have sub-resources. Complex data models are, as such, harder to model with LWM2M.

In situations where resources need to be modified, CoMI uses the CoAP PATCH operation when resources are modified partially. However, LWM2M uses the CoAP PUT and POST operations, even when a subset of the resource needs modifications.

Authors' Addresses

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com

CoRE
Internet-Draft
Intended status: Standards Track
Expires: July 22, 2017

P. van der Stok
consultant
A. Bierman
YumaWorks
M. Veillette
Trilliant Networks Inc.
A. Pelov
Acklio
January 18, 2017

CoAP Management Interface
draft-vanderstok-core-comi-11

Abstract

This document describes a network management interface for constrained devices and networks, called CoAP Management Interface (CoMI). The Constrained Application Protocol (CoAP) is used to access data resources specified in YANG, or SMIV2 converted to YANG. CoMI uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction. CoMI extends the set of YANG based protocols, NETCONF and RESTCONF, with the capability to manage constrained devices and networks.

Note

Discussion and suggestions for improvement are requested, and should be sent to core@ietf.org.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 22, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. CoMI Architecture	5
2.1. Major differences between RESTCONF and CoMI	7
2.2. Compression of YANG identifiers	7
3. Example syntax	8
4. CoAP Interface	8
5. /c Function Set	9
5.1. Using the 'k' query parameter	10
5.2. Data Retrieval	11
5.2.1. Using the 'c' query parameter	12
5.2.2. Using the 'd' query parameter	12
5.2.3. GET	13
5.2.4. FETCH	15
5.3. Data Editing	16
5.3.1. Data Ordering	16
5.3.2. POST	16
5.3.3. PUT	17
5.3.4. iPATCH	18
5.3.5. DELETE	19
5.4. Full Data Store access	19
5.4.1. Full Data Store examples	20
5.5. Notify functions	21
5.5.1. Notify Examples	22
5.6. RPC statements	22
5.6.1. RPC Example	23
6. Access to MIB Data	23
7. Use of Block	25
8. Resource Discovery	25
9. Error Handling	27
10. Security Considerations	28

11. IANA Considerations	29
12. Acknowledgements	29
13. Changelog	30
14. References	33
14.1. Normative References	33
14.2. Informative References	35
Appendix A. YANG example specifications	37
A.1. ietf-system	37
A.2. server list	38
A.3. interfaces	39
A.4. Example-port	40
A.5. IP-MIB	41
Appendix B. Comparison with LWM2M	43
B.1. Introduction	43
B.2. Defining Management Resources	44
B.3. Identifying Management Resources	44
B.4. Encoding of Management Resources	45
Authors' Addresses	45

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. The messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

This draft describes the CoAP Management Interface which uses CoAP methods to access structured data defined in YANG [RFC7950]. This draft is complementary to the draft [I-D.ietf-netconf-restconf] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data sets, specified in a standardized language such as YANG, promotes interoperability between devices and applications from different manufacturers. A large amount of Management Information Base (MIB) [mibreg] specifications already exists for monitoring purposes. This data can be accessed in RESTCONF or CoMI if the server converts the SMIV2 modules to YANG, using the mapping rules defined in [RFC6643].

CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips.

To promote small packets, CoMI uses a YANG to CBOR mapping [I-D.ietf-core-yang-cbor] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Readers of this specification should be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

The following terms are defined in the NETCONF protocol [RFC6241]: client, configuration data, datastore, and server.

The following terms are defined in the YANG data modelling language [RFC7950]: anydata, anyxml, container, data node, key, key leaf, leaf, leaf-list, and list.

The following terms are defined in RESTCONF protocol [I-D.ietf-netconf-restconf]: data resource, datastore resource, edit operation, query parameter, and target resource.

The following terms are defined in this document:

data node instance: An instance of a data node specified in a YANG module present in the server. The instance is stored in the memory of the server.

Notification instance: An instance of a schema node of type notification, specified in a YANG module present in the server. The instance is generated in the server at the occurrence of the corresponding event and appended to a stream.

YANG schema item identifier: Numeric identifier which replaces the name identifying a YANG item (see section 6.2 of [RFC7950]) (anydata, anyxml, data node, RPC, Action, Notification, Identity, Module name, Submodule name, Feature).

list instance identifier: Handle used to identify a YANG data node that is an instance of a YANG "list" specified with the values of the key leaves of the list.

single instance identifier: Handle used to identify a specific data node which can be instantiated only once. This includes data nodes defined at the root of a YANG module or submodule and data

nodes defined within a container. This excludes data nodes defined within a list or any children of these data nodes.

instance identifier: List instance identifier or single instance identifier.

data node value: Value assigned to a data node instance. Data node values are encoded based on the rules defined in section 4 of [I-D.ietf-core-yang-cbor].

set of data node instances: Represents the payload of CoAP methods when a collection is sent or returned. There are two possibilities, dependent on Request context:

1. CBOR array of pair(s) <instance identifier, data node value >
2. CBOR map of pair(s) <instance identifier, data node value >

TODO: Reduce to one, if possible

The following list contains the abbreviations used in this document.

SID: YANG Schema Item iDentifier.

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for the reading and modifying the content of a datastore used for the management of the instrumented node.

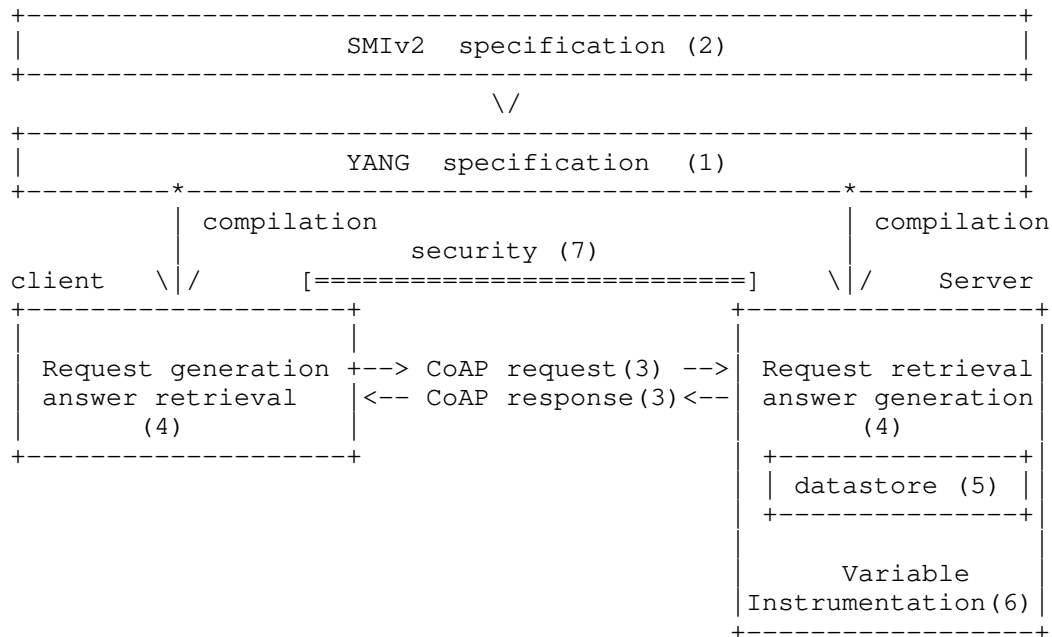


Figure 1: Abstract CoMI architecture

Figure 1 is a high level representation of the main elements of the CoAP management architecture. A client sends requests as payload in packets over the network to a managed constrained node.

The different numbered components of Figure 1 are discussed according to component number.

- (1) YANG specification: contains a set of named and versioned modules.
- (2) SMIv2 specification: A named module specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIv2 specifications to YANG specifications.
- (3) CoMI messages: The CoMI client sends request messages to and receives response messages from the CoMI server.
- (4) Retrieval, generation: The server and client parse the CoMI request/response and identify the corresponding instances in the datastore based on YANG specification.

- (5) Datastore: The store is composed of two parts: Operational state and Configuration datastore. Datastore also supports RPCs and event streams.
- (6) Variable instrumentation: This code depends on implementation of drivers and other node specific aspects.
- (7) Security: The server MUST prevent unauthorized users from reading or writing any data resources. CoMI relies on security protocols such as DTLS [RFC6347] to secure CoAP communication.

2.1. Major differences between RESTCONF and CoMI

CoMI uses CoAP/UDP as transport protocol and CBOR as payload format [I-D.ietf-core-yang-cbor]. RESTCONF uses HTTP/TCP as transport protocol and JSON [RFC7159] or XML [XML] as payload formats. CoMI encodes YANG identifier strings as numbers, where RESTCONF does not.

CoMI uses the methods FETCH and iPATCH, not used by RESTCONF. RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not used by CoMI.

CoMI servers cannot change the order of user-ordered data. CoMI does not support insert-mode (first, last, before, after) and insertion-point (before, after) which are supported by RESTCONF. Many CoAP servers will not support date and time functions. For that reason CoMI does not support the start, stop options for events.

CoMI servers only implement the efficient "trim" mode for default values.

The CoMI servers do not support the following RESTCONF functionality:

- o The "fields" query parameter to query multiple instances.
- o The 'filter' query that involves XML parsing, 'content', and 'depth', query parameters.

2.2. Compression of YANG identifiers

In the YANG specification items are identified with a name string. In order to significantly reduce the size of identifiers used in CoMI, numeric object identifiers are used instead of these strings. The specific encoding of the object identifiers is not hard-wired in the protocol.

Examples of object identifier encoding formats are described in [I-D.ietf-core-sid].

3. Example syntax

This section presents the notation used for the examples. The YANG specifications that are used throughout this document are shown in Appendix A. The example specifications are taken over from existing modules and annotated with SIDs. The values of the SIDs are taken over from [yang-cbor].

CBOR is used to encode CoMI request- and response- payloads. The CBOR syntax of the YANG payloads is specified in [RFC7049]. The payload examples are notated in Diagnostic notation (defined in section 6 of [RFC7049]) that can be automatically converted to CBOR.

A YANG (item identifier, item value) pair is mapped to a CBOR (key, value) pair. The YANG item value is encoded as specified in [I-D.ietf-core-yang-cbor]. The YANG item identifier can be a SID (single node identifier) or a CBOR array with the structure [SID, key1, key2] (list node identifier), where SID is a list identifier and the key values specify the list instance. The YANG item value can be any CBOR major type.

Delta encoding is used for the SIDs. The notation +n is used when the SID has the value PREC+n where PREC is the SID of the parent container, or PREC is the SID of the preceding entity in a CBOR array.

In all examples the resource path in the URI is expressed as a SID, represented as a base64 number. SIDs in the payload are represented as decimal numbers.

4. CoAP Interface

In CoAP a group of links can constitute a Function Set.

TODO: what will happen to term Function Set ?

The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Function Set. CoMI end-points that implement the CoMI management protocol support at least one discoverable management resource of resource type (rt): core.c, with path: /c, where c is short-hand for CoMI. The path root /c is recommended but not compulsory (see Section 8).

The path prefix /c has resources accessible with the following three paths:

/c: YANG-based data with path "/c" and using CBOR content encoding format. This path represents a datastore resource which contains

YANG data resources as its descendant nodes. The data nodes are identified with their SID with format /c/SID.

/c/mod.uri: URI identifying the location of the server module information, with path "/c/mod.uri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG encoded values. An Entity Tag MUST be maintained for this resource by the server, which MUST be changed to a new value when the set of YANG modules in use by the server changes.

/c/s: String identifying the default stream resource to which YANG notification instances are appended. Notification support is optional, so this resource will not exist if the server does not support any notifications.

The mapping of YANG data node instances to CoMI resources is as follows: A YANG module describes a set of data trees composed of YANG data nodes. Every data node of the YANG modules loaded in the CoMI server represents a resource of the datastore container (e.g. /c/<sid>

When multiple instances of a list node exist, instance selection is possible as described in Section 5.2.4 and Section 5.2.3.1.

TODO; reference to fetch and patch content formats.

The profile of the management function set, with IF=core.c, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

name	path	rt	Data Type
Management	/c	core.c	n/a
Data	/c	core.c.data	application/cbor
Module Set URI	/c/mod.uri	core.c.moduri	application/cbor
Events	/c/s	core.c.stream	application/cbor

5. /c Function Set

The /c Function Set provides a CoAP interface to manage YANG servers.

The methods used by CoMI are:

Operation	Description
GET	Retrieve the datastore resource or a data resource
FETCH	Retrieve (partial) data resource(s)
POST	Create a data resource, invoke RPC
PUT	Create or replace a data resource
iPATCH	Idem-potently create, replace, and delete data resource(s) (partially)
DELETE	Delete a data resource

There is one query parameters for the GET, PUT, POST, and DELETE methods.

Query Parameter	Description
k	Select an instance of a list node

This parameter is not used for FETCH and iPATCH, because their request payloads support list instance selection.

5.1. Using the 'k' query parameter

The "k" (key) parameter specifies the instance of a list node. The SID in the URI is followed by the (?k=key1, key2,...). Where SID identifies a list node, and key1, key2 are the values of the key leaves that specify an instance of the list.

Key values are encoded using the rules defined in the following table:

YANG datatype	Binary representation	Text representation
uint8,uint16,uint32, uint64	CBOR unsigned integer	int_to_text(number)
int8, int16,int32, int64	CBOR negative integer	base64 (CBOR representation)
decimal64	CBOR decimal fractions	base64 (CBOR representation)
string	CBOR text or string	text
boolean	CBOR false or true	"0" or "1"
enumeration	CBOR unsigned integer	int_to_text (number)
bits	CBOR byte string	base64 (CBOR representation)
binary	CBOR byte string	base64 (binary value)
identityref	CBOR unsigned integer	int_to_text (number)
union		base64 (CBOR representation)
List instance identifier	CBOR unsigned integer	base64 (CBOR representation)
List instance identifier	CBOR array	Base64 (CBOR representation)

5.2. Data Retrieval

One or more data node instances can be retrieved by the client. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252] and to the FETCH method defined in section 2 of [I-D.ietf-core-etch].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [RFC7959] is used, as explained in more detail in Section 7.

CoMI uses the FETCH payload for retrieving a subset of the datastore.

There are two additional query parameters for the GET and FETCH methods.

Query Parameter	Description
c	Control selection of configuration and non-configuration data nodes (GET and FETCH)
d	Control retrieval of default values.

5.2.1. Using the 'c' query parameter

The 'c' (content) parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
c	Return only configuration descendant data nodes
n	Return only non-configuration descendant data nodes
a	Return all descendant data nodes

This parameter is only allowed for GET and FETCH methods on datastore and data resources. A 4.00 Bad Request error is returned if used for other methods or resource types.

If this query parameter is not present, the default value is "a".

5.2.2. Using the 'd' query parameter

The "d" (with-defaults) parameter controls how the default values of the descendant nodes of the requested data nodes will be processed.

The allowed values are:

Value	Description
a	All data nodes are reported Defined as 'report-all' in section 3.1 of [RFC6243].
t	Data nodes set to the YANG default are not reported. Defined as 'trim' in section 3.2 of [RFC6243].

If the target of a GET or FETCH method is a data node that represents a leaf that has a default value, and the leaf has not been given a value yet, the server MUST return the leaf.

If the target of a GET method is a data node that represents a container or list that has any child resources with default values, for the child resources that have not been given value yet, the server MUST not return the child resource if this query parameter is set to 't' and MUST return the child resource if this query parameter is set to 'a'.

If this query parameter is not present, the default value is 't'.

5.2.3. GET

A request to read the values of a data node instance is sent with a confirmable CoAP GET message. A single instance identifier is specified in the URI path prefixed with /c.

FORMAT:

```
GET /c/<instance identifier>
```

```
2.05 Content (Content-Format: application/cbor)
<data node value>
```

The returned payload is composed of all the children associated with the specified data node instance.

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.2.3.1. GET Examples

Using for example the current-datetime leaf from Appendix A.1, a request is sent to retrieve the value of system-state/clock/current-datetime specified in container system-state. The ID of system-

state/clock/current-datetime is 1719, encoded in base64 this yields a3. The answer to the request returns a <value>, transported as a single CBOR string item.

REQ: GET example.com/c/a3

RES: 2.05 Content (Content-Format: application/cbor)
"2014-10-26T12:16:31Z"

For example, the GET of the clock node (ID = 1717; base64: a1), sent by the client, results in the following returned value sent by the server, transported as a CBOR map containing 2 pairs:

REQ: GET example.com/c/a1

RES: 2.05 Content (Content-Format: application/cbor)
{
 +2 : "2014-10-26T12:16:51Z", / ID 1719 /
 +1 : "2014-10-21T03:00:00Z" / ID 1718 /
}

A "list" node can have multiple instances. Accordingly, the returned payload of GET is composed of all the instances associated with the selected list node.

For example, look at the example in Appendix A.3. The GET of the /interfaces/interface/ (with identifier 1533, base64: X9) results in the following returned payload, transported as a CBOR array with 2 elements.

REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/cbor)

```
[
  {+4 : "eth0",           / name (ID 1537) /
    +1 : "Ethernet adaptor", / description (ID 1534) /
    +5 : 1179,             / type, (ID 1538) identity /
                                / ethernetCsmacd (ID 1179) /
    +2 : true              / enabled ( ID 1535) /
  },
  {+4 : "eth1",           / name (ID 1537) /
    +1 : "Ethernet adaptor", / description (ID 1534) /
    +5 : 1179,             / type, (ID 1538) identity /
                                / ethernetCsmacd (ID 1179) /
    +2 : false            / enabled /
  }
]
```

It is equally possible to select a leaf of one instance of a list or a complete instance container with GET. The instance identifier is the numeric identifier of the list followed by the specification of the values for the key leaves that uniquely identify the list instance. The instance identifier looks like: SID?k=key-value. The key of "interface" is the "name" leaf. The example below requests the description leaf of the instance with name="eth0" (ID=1534, base64: X-). The value of the description leaf is returned.

REQ: GET example.com/c/X-?k="eth0"

RES: 2.05 Content (Content-Format: application/cbor)
"Ethernet adaptor"

5.2.4. FETCH

The FETCH is used to retrieve a list of data node values. The FETCH Request payload contains a CBOR list of instance identifiers.

FORMAT:

```
FETCH /c/ Content-Format (application/YANG-fetch+cbor)
<CBOR array of instance identifiers>
```

```
2.05 Content (Content-Format: application/YANG-patch+cbor)
<CBOR array of data node values>
```

The instance identifier is a SID or a CBOR array containing the SID followed by key values that identify the list instance (sec 5.13.1 of [I-D.ietf-core-yang-cbor]). In the payload of the returned data node

values, delta encoding is used as described in [I-D.ietf-core-yang-cbor].

5.2.4.1. FETCH examples

The example uses the current-datetime leaf and the interface list from Appendix A.1. In the following example the value of current-datetime (ID 1719) and the interface list (ID 1533) instance identified with name="eth0" are queried.

```
REQ:  FETCH /c Content-Format (application/YANG-fetch+cbor)
      [ 1719,                / ID 1719 /
        [-186, "eth0"]      / ID 1533 with name = "eth0" /
      ]
```

```
RES:  2.05 Content Content-Format (application/YANG-patch+cbor)
      [
        "2014-10-26T12:16:31Z",
        {
          +4 : "eth0",          / name (ID 1537) /
          +1 : "Ethernet adaptor", / description (ID 1534) /
          +5 : 1179,            / type (ID 1538), identity /
                                / ethernetCsmacd (ID 1179) /
          +2 : true              / enabled (ID 1535) /
        }
      ]
```

TODO: align with future FETCH content format.

5.3. Data Editing

CoMI allows datastore contents to be created, modified and deleted using CoAP methods.

5.3.1. Data Ordering

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as arrays so messages will preserve their order.

5.3.2. POST

Data resources are created with the POST method. The CoAP POST operation is used in CoMI for creation of data resources and the invocation of "ACTION" and "RPC" resources. Refer to Section 5.6 for details on "ACTION" and "RPC" resources.

A request to create the values of an instance of a container or leaf is sent with a confirmable CoAP POST message. A single SID is specified in the URI path prefixed with /c.

FORMAT:

```
POST /c/<instance identifier> Content-Format(application/cbor)
<data node value>
```

2.01 Created (Content-Format: application/cbor)

If the data resource already exists, then the POST request MUST fail and a "4.09 Conflict" status-line MUST be returned

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.2.1. Post example

The example uses the interface list from Appendix A.1. Example is creating a new version of the container interface (ID = 1533):

REQ: POST /c/X9 Content-Format(application/cbor)

```
{
  +4 : "eth0",           / name (ID 1537) /
  +1 : "Ethernet adaptor", / description (ID 1534) /
  +5 : 1179,             / type (ID 1538), identity /
                           / ethernetCsmacd (ID 1179) /
  +2 : true               / enabled (ID 1535) /
}
```

RES: 2.01 Created (Content-Format: application/cbor)

5.3.3. PUT

Data resource instances are created or replaced with the PUT method. The PUT operation is supported in CoMI. A request to set the value of a data node instance is sent with a confirmable CoAP PUT message.

FORMAT:

```
PUT /c/<instance identifier> Content-Format(application/cbor)
<data node value>
```

2.01 Created

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.3.1. PUT example

The example uses the interface list from Appendix A.1. Example is renewing an instance of the list interface (ID = 1533) with key name="eth0":

```
REQ: PUT /c/X9?k="eth0" Content-Format(application/cbor)
{
  +4 : "eth0",           / name (ID 1537) /
  +1 : "Ethernet adaptor", / description (ID 1534) /
  +5 : 1179,             / type (ID 1538), identity /
                           / ethernetCsmacd ( ID 1179) /
  +2 : true              / enabled (ID 1535) /
}
RES: 2.04 Changed
```

5.3.4. iPATCH

One or multiple data resource instances are replaced with the idempotent iPATCH method [I-D.ietf-core-etch]. A request is sent with a confirmable CoAP iPATCH message.

There are no query parameters for the iPATCH method.

The processing of the iPATCH command is specified by the CBOR payload. The CBOR patch payload describes the changes to be made to target YANG data nodes [I-D.bormann-appsawg-cbor-merge-patch]. If the CBOR patch payload contains data node instances that are not present in the target, these instances are added or silently ignored dependent of the payload information. If the target contains the specified instance, the contents of the instances are replaced with the values of the payload. Null values indicate the removal of existing values.

```
FORMAT:
  iPATCH /c Content-Format(application/YANG-patch+cbor)
  <set of data node instances>
```

2.04 Changed

5.3.4.1. iPATCH example

The example uses the interface list from Appendix A.3, and the timezone-utc-offset leaf from Appendix A.1. In the example one leaf (timezone-utc-offset) and one container (interface) instance are changed.

```
REQ: iPATCH /c Content-Format(application/YANG-patch+cbor)
[
  [1533, "eth0"] ,                / interface (ID = 1533) /
  {
    +4 : "eth0",                  / name (ID 1537) /
    +1 : "Ethernet adaptor",      / description (ID 1534) /
    +5 : 1179,                    / type (ID 1538), identity /
                                / ethernetCsmacd (ID 1179) /
    +2 : true                      / enabled (ID 1535) /
  },
  +203 , 60                       / timezone-utc-offset (delta = 1736-1533) /
]
```

RES: 2.04 Changed

TODO: Align with future cbor-merge-patch content format.

5.3.5. DELETE

Data resource instances are deleted with the DELETE method. The RESTCONF DELETE operation is supported in CoMI.

FORMAT:

Delete /c/<instance identifier>

2.02 Deleted

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.5.1. DELETE example

The example uses the interface list from Appendix A.3. Example is deleting an instance of the container interface (ID = 1533):

REQ: DELETE /c/X9?k="eth0"

RES: 2.02 Deleted

5.4. Full Data Store access

The methods GET, PUT, POST, and DELETE can be used to return, replace, create, and delete the whole data store respectively.

FORMAT:

```
GET /c
2.05 Content (Content-Format: application/cbor)
<array of data node instances>

PUT /c
(Content-Format: application/cbor)
<array of data node instances>
2.04 Changed

POST /c
(Content-Format: application/cbor)
<array of data node instances>
2.01 Created

DELETE /c
2.02 Deleted
```

The array of data node instances represents an array of all root nodes in the data store after the PUT, POST and GET method invocations.

5.4.1. Full Data Store examples

The example uses the interface list and the clock container from Appendix A.3. Assume that the data store contains two root objects: the list interface (ID 1533) with one instance and the container Clock (ID 1717). After invocation of GET an array with these two objects is returned:

```

RQ:  GET /c
RES:  2.05 Content Content-Format (application/YANG-patch+cbor)
[
  {1717:
    { +2: "2016-10-26T12:16:31Z", / current-datetime (ID 1719) /
      +1: "2014-10-05T09:00:00Z" / boot-datetime (ID 1718) /
    },
  -186: / clock (ID 1533) /
    {
      +4 : "eth0", / name (ID 1537) /
      +1 : "Ethernet adaptor", / description (ID 1534) /
      +5 : 1179, / type (ID 1538), identity: /
        / ethernetCsmacd (ID 1179) /
      +2 : true / enabled (ID 1535) /
    }
  }
]

```

5.5. Notify functions

Notification by the server to a selection of clients when an event occurs in the server is an essential function for the management of servers. CoMI allows events specified in YANG [RFC5277] to be notified to a selection of requesting clients. The server appends newly generated events to a stream. There is one, so-called "default", stream in a CoMI server. The /c/s resource identifies the default stream. The server MAY create additional stream resources. When a CoMI server generates an internal event, it is appended to the chosen stream, and the content of a notification instance is ready to be sent to all CoMI clients which observe the chosen stream resource.

Reception of generated notification instances is enabled with the CoAP Observe [RFC7641] function. The client subscribes to the notifications by sending a GET request with an "Observe" option, specifying the /c/s resource when the default stream is selected.

Every time an event is generated, the chosen stream is cleared, and the generated notification instance is appended to the chosen stream(s). After appending the instance, the contents of the instance is sent to all clients observing the modified stream.

FORMAT:

```

  Get /<stream-resource>
      Content-Format(application/YANG-patch+cbor) Observe(0)

2.05 Content Content-Format(application/YANG-patch+cbor)
<set of data node instances>

```

5.5.1. Notify Examples

Suppose the server generates the event specified in Appendix A.4. By executing a GET on the /c/s resource the client receives the following response:

REQ: GET /c/s Observe(0) Token(0x93)

RES: 2.05 Content Content-Format(application/YANG-patch+cbor)
Observe(12) Token(0x93)

```
{
  60010 :                / example-port-fault (ID 60010) /
  {
    +1 : "0/4/21",        / port-name (ID 60011) /
    +2 : "Open pin 2"     / port-fault (ID 60012) /
  },
  60010 :                / example-port-fault (ID 60010) /
  {
    +1 : "1/4/21",        / port-name (ID 60011) /
    +2 : "Open pin 5"     / port-fault (ID 60012) /
  }
}
```

In the example, the request returns a success response with the contents of the last two generated events. Consecutively the server will regularly notify the client when a new event is generated.

To check that the client is still alive, the server MUST send confirmable notifications once in a while. When the client does not confirm the notification from the server, the server will remove the client from the list of observers [RFC7641].

5.6. RPC statements

The YANG "action" and "RPC" statements specify the execution of a Remote procedure Call (RPC) in the server. It is invoked using a POST method to an "Action" or "RPC" resource instance. The Request payload contains the values assigned to the input container when specified with the action station. The Response payload contains the values of the output container when specified.

The returned success response code is 2.05 Content.

FORMAT:

```
POST /c/<instance identifier>
      Content-Format(application/YANG-patch+cbor)
<input node value>

2.05 Content Content-Format (application/YANG-patch+cbor)
<output node value>
```

There "k" query parameter is allowed for the POST method when used for an action invocation.

5.6.1. RPC Example

The example is based on the YANG action specification of Appendix A.2. A server list is specified and the action "reset" (ID 60002, base64: Opq), that is part of a "server instance" with key value "myserver", is invoked.

```
REQ:  POST /c/Opq?k="myserver"
      Content-Format(application/YANG-patch+cbor)
{
  +1 : "2016-02-08T14:10:08Z09:00" / reset-at (ID 60003) /
}

RES:  2.05 Content Content-Format(application/YANG-patch+cbor)
{
  +2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (ID 60004)/
}
```

6. Access to MIB Data

Appendix A.5 shows a YANG module mapped from the SMI specification "IP-MIB" [RFC4293]. The following example shows the "ipNetToPhysicalEntry" list with 2 instances, using diagnostic notation without delta encoding.

```

{
  60021 :                               / list ipNetToPhysicalEntry /
  [
    {
      60022 : 1,                        / ipNetToPhysicalIfIndex /
      60023 : 1,                        / ipNetToPhysicalNetAddressType: ipv4 /
      60024 : h'0A000033',              / ipNetToPhysicalNetAddress /
      60025 : h'00000A01172D',          / ipNetToPhysicalPhysAddress /
      60026 : 2333943,                  / ipNetToPhysicalLastUpdated /
      60027 : 4,                        / ipNetToPhysicalType: static /
      60028 : 1,                        / ipNetToPhysicalState: reachable /
      60029 : 1                          / ipNetToPhysicalRowStatus: active /
    },
    {
      60022 : 1,                        / ipNetToPhysicalIfIndex /
      60023 : 1,                        / ipNetToPhysicalNetAddressType: ipv4 /
      60024 : h'09020304',              / ipNetToPhysicalNetAddress /
      60025 : h'00000A36200A',          / ipNetToPhysicalPhysAddress /
      60026 : 2329836,                  / ipNetToPhysicalLastUpdated /
      60027 : 3,                        / ipNetToPhysicalType: dynamic /
      60028 : 6,                        / ipNetToPhysicalState: unknown /
      60029 : 1                          / ipNetToPhysicalRowStatus: active /
    }
  ]
}

```

The IPv4 addresses A.0.0.33 and 9.2.3.4 are encoded in CBOR as h'0A000033' and h'09020304' respectively. In the following example exactly one instance is requested from the ipNetToPhysicalEntry (ID 60021, base64: Oz1). The h'09020304' value is encoded in base64 as AJAgME.

In this example one instance of /ip/ipNetToPhysicalEntry that matches the keys ipNetToPhysicalIfIndex = 1, ipNetToPhysicalNetAddressType = ipv4 and ipNetToPhysicalNetAddress = 9.2.3.4 (h'09020304', base64: AJAgME).

```
REQ: GET example.com/c/Oz1?k="1,1,AJAgME"
```

```
RES: 2.05 Content (Content-Format: application/YANG-patch+cbor)
{
  +1 : 1,                / ( SID 60022 ) /
  +2 : 1,                / ( SID 60023 ) /
  +3 : h'09020304',      / ( SID 60024 ) /
  +4 : h'00000A36200A',  / ( SID 60025 ) /
  +5 : 2329836,          / ( SID 60026 ) /
  +6 : 3,                / ( SID 60027 ) /
  +7 : 6,                / ( SID 60028 ) /
  +8 : 1                 / ( SID 60029 ) /
}
```

7. Use of Block

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of text need to be transported the datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [RFC7959] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying transport sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

Beware of race conditions. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. In the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the variables, the transported results do not correspond with a state having occurred in the server; or worse the returned values are inconsistent. For example: array length does not correspond with actual number of items. It may be advisable to use CBOR maps or CBOR arrays of undefined length which are foreseen for data streaming purposes.

8. Resource Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to `"/.well-known/core"` including a resource type (RT) parameter with the value `"core.c"` [RFC6690].

Upon success, the return payload will contain the root resource of the management data. It is up to the implementation to choose its root resource, but it is recommended that the value `"/c"` is used, where possible. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.c
```

```
RES: 2.05 Content </c>; rt="core.c"
```

Management objects MAY be discovered with the standard CoAP resource discovery. The implementation can add the encoded values of the object identifiers to `/.well-known/core` with `rt="core.c.data"`. The available objects identified by the encoded values can be discovered by sending a GET request to `"/.well-known/core"` including a resource type (RT) parameter with the value `"core.c.data"`. Upon success, the return payload will contain the registered encoded values and their location. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.c.data
```

```
RES: 2.05 Content </c/BaAiN>; rt="core.c.data",  
</c/CF_fA>; rt="core.c.data"
```

Lists of encoded values may become prohibitively long. It is discouraged to provide long lists of objects on discovery. Therefore, it is recommended that details about management objects are discovered by reading the YANG module information stored in for example the `"ietf-comi-yang-library"` module [I-D.veillette-core-cool-library]. The resource `"/c/mod.uri"` is used to retrieve the location of the YANG module library.

The module list can be stored locally on each server, or remotely on a different server. The latter is advised when the deployment of many servers are identical.

Local in example.com server:

REQ: GET example.com/c/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
 "mod.uri" : "example.com/c/modules"
}

Remote in example-remote-server:

REQ: GET example.com/c/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
 "moduri" : "example-remote-server.com/c/group17/modules"
}

Within the YANG module library all information about the module is stored such as: module identifier, identifier hierarchy, grouping, features and revision numbers.

9. Error Handling

In case a request is received which cannot be processed properly, the CoMI server MUST return an error message. This error message MUST contain a CoAP 4.xx or 5.xx response code, and SHOULD include additional information in the payload.

Such an error message payload is a text string, using the following structure:

CoMI error: xxxx "error text"

The characters xxxx represent one of the values from the table below, and the OPTIONAL "error text" field contains a human readable explanation of the error.

CoMI Error Code	CoAP Error Code	Description
0	4.xx	General error
1	4.13	Request too big
2	4.00	Response too big
3	4.00	Unknown identifier
4	4.00	Invalid value
5	4.05	Attempt to write read-only variable
6	5.01	No access
7	4.00	Wrong type
8	4.15	Unknown encoding
9	4.0	Wrong value
10	4.0	Not created
11	4.04	Resource unavailable
12	4.01	Authorization error
13	4.0	Bad attribute
14	4.0	Unknown attribute
15	4.0	Missing attribute

The CoMI error codes are motivated by the error-status values defined in [RFC3416], and the error tags defined in [I-D.ietf-netconf-restconf].

10. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. This requires integrity protection of both requests and responses, and depending on the application encryption.

CoMI re-uses the security mechanisms already available to CoAP as much as possible. This includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorization mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorization may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However some adaptations may still be required, to cater for CoMI's specific requirements.

11. IANA Considerations

'rt="core.c"' needs registration with IANA.

'rt="core.c.data"' needs registration with IANA.

'rt="core.c.moduri"' needs registration with IANA.

'rt="core.c.stream"' needs registration with IANA.

Content types to be registered:

- o application/YANG-patch+cbor
- o application/YANG-fetch+cbor

12. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification and specified CBOR encoding and use of hashes.

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR.

Timothy Carey has provided the text for Appendix B.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Juergen

Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

13. Changelog

Changes from version 00 to version 01

- o Focus on MIB only
- o Introduced CBOR, JSON, removed BER
- o defined mappings from SMI to xx
- o Introduced the concept of addressable table rows

Changes from version 01 to version 02

- o Focus on CBOR, used JSON for examples, removed XML and EXI
- o added uri-query attributes mod and con to specify modules and contexts
- o Definition of CBOR string conversion tables for data reduction
- o use of Block for multiple fragments
- o Error returns generalized
- o SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- o Added security considerations

Changes from version 03 to version 04

- o Added design considerations section
- o Extended comparison of management protocols in introduction
- o Added automatic generation of CBOR tables
- o Moved lowpan table to Appendix

Changes from version 04 to version 05

- o Merged SNMP access with RESTCONF access to management objects in small devices

- o Added CoMI architecture section
- o Added RESTCONF NETMOD description
- o Rewrote section 5 with YANG examples
- o Added server and payload size appendix
- o Removed Appendix C for now. It will be replaced with a YANG example.

Changes from version 04 to version 05

- o Extended examples with hash representation
- o Added keys query parameter text
- o Added select query parameter text
- o Better separation between specification and instance
- o Section on discovery updated
- o Text on rehashing introduced
- o Elaborated SMI MIB example
- o YANG library use described
- o use of BigEndian/LittleEndian in Hash generation specified

Changes from version 05 to version 06

- o Hash values in payload as hexadecimal and in URL in base64 numbers
- o Streamlined CoMI architecture text
- o Added select query parameter text
- o Data editing optional
- o Text on Notify added
- o Text on rehashing improved with example

Changes from version 06 to version 07

- o reduced payload size by removing JSON hierarchy

- o changed rehash handling to support small clients
- o added LWM2M comparison
- o Notification handling as specified in YANG
- o Added Patch function
- o Rehashing completely reviewed
- o Discover type of YANG name encoding
- o Added new resource types
- o Read-only servers introduced
- o Multiple updates explained

Changes from version 07 to version 08

- o Changed YANG Hash algorithm to use modulename instead of prefix
- o Added rehash bit to allow return values to identify rehashed nodes in the response
- o Removed /c/mod.set resource since this is not needed
- o Clarified that YANG Hash is done even for unimplemented objects
- o YANG lists transported as CBOR maps of maps
- o Adapted examples with more CBOR explanation
- o Added CBOR code examples in new appendix
- o Possibility to use other than default stream
- o Added text and examples for Patch payload
- o Repaired some examples
- o Added appendices on hash clash probability and hash clash storage overhead

Changes from version 08 to version 09

- o Removed hash and YANG to CBOR sections

- o removed hashes from examples.
- o Added RPC
- o Added content query parameter.
- o Added default handling.
- o Listed differences with RESTCONF

Changes from version 09 to version 10. This is the merge of cool-01 with comi-09.

- o Merged with CoOL SIDs
- o Introduced iPATCH, PATCH and FETCH
- o Update of LWM2M comparison
- o Added appendix with module examples
- o Removed introductory text
- o Removed references

Changes from version 10 to version 11

- o Introduction streamlined
- o Error codes streamlined
- o Examples updated to latest SID values
- o Update of the YANG specifications in the appendix

14. References

14.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<http://www.rfc-editor.org/info/rfc5277>>.

- [RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <<http://www.rfc-editor.org/info/rfc6243>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<http://www.rfc-editor.org/info/rfc7959>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.
- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.
- [I-D.ietf-core-etch]
Stok, P., Bormann, C., and A. Sehgal, "Patch and Fetch Methods for Constrained Application Protocol (CoAP)", draft-ietf-core-etch-04 (work in progress), November 2016.
- [I-D.bormann-appsawg-cbor-merge-patch]
Bormann, C. and P. Stok, "CBOR Merge Patch", draft-bormann-appsawg-cbor-merge-patch-00 (work in progress), March 2016.
- [I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-03 (work in progress), October 2016.

14.2. Informative References

- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <<http://www.rfc-editor.org/info/rfc2578>>.
- [RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <<http://www.rfc-editor.org/info/rfc3410>>.
- [RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002, <<http://www.rfc-editor.org/info/rfc3416>>.
- [RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <<http://www.rfc-editor.org/info/rfc4293>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <<http://www.rfc-editor.org/info/rfc6643>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., Koster, M., and C. Groves, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-07 (work in progress), December 2016.

[I-D.ietf-core-sid]
Somaraju, A., Veillette, M., Pelov, A., Turner, R., and A. Minaburo, "YANG Schema Item Identifier (SID)", draft-ietf-core-sid-00 (work in progress), October 2016.

[I-D.veillette-core-cool]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "Constrained Objects Language", draft-veillette-core-cool-02 (work in progress), July 2016.

[I-D.veillette-core-cool-library]
Veillette, M., "Constrained YANG Module Library", draft-veillette-core-cool-library-00 (work in progress), August 2016.

[XML] "Extensible Markup Language (XML)",
Web <http://www.w3.org/xml>.

[OMA] "OMA-TS-LightweightM2M-V1_0-20131210-C", Web
http://technical.openmobilealliance.org/Technical/current_releases.aspx.

[OMNA] "Open Mobile Naming Authority (OMNA)", Web
<http://http://technical.openmobilealliance.org/Technical/technical-information/omna>.

[netconfcentral]
"NETCONF Central: library of YANG modules",
Web <http://www.netconfcentral.org/modulelist>.

[mibreg] "Structure of Management Information (SMI) Numbers (MIB Module Registrations)", Web
<http://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml/>.

[yang-cbor]
"yang-cbor Registry", Web <https://github.com/core-wg/yang-cbor/tree/master/registry/>.

Appendix A. YANG example specifications

This appendix shows 5 YANG example specifications taken over from as many existing YANG modules. The YANG modules are available from [netconfcentral]. Each YANG item identifier is accompanied by its SID shown after the "//" comment sign, taken from [yang-cbor].

A.1. ietf-system

Excerpt of the YANG module ietf-system [RFC7317].

```
module ietf-system {
  container system {                               // SID 1715
    container clock {                               // SID 1734
      choice timezone {
        case timezone-name {
          leaf timezone-name {                     // SID 1735
            type timezone-name;
          }
        }
        case timezone-utc-offset {
          leaf timezone-utc-offset {               // SID 1736
            type int16 {
            }
          }
        }
      }
    }
  }
  container ntp {                                  // SID 1750
    leaf enabled {                                  // SID 1751
      type boolean;
      default true;
    }
    list server {                                   // SID 1752
      key name;
      leaf name {                                   // SID 1755
        type string;
      }
    }
    choice transport {
      case udp {
        container udp {                             // SID 1757
          leaf address {                             // SID 1758
            type inet:host;
          }
          leaf port {                               // SID 1759
            type inet:port-number;
          }
        }
      }
    }
  }
}
```

```
    }
  }
  leaf association-type {           // SID 1753
    type enumeration {
      enum server {
      }
      enum peer {
      }
      enum pool {
      }
    }
  }
  leaf iburst {                     // SID 1754
    type boolean;
  }
  leaf prefer {                     // SID 1756
    type boolean;
    default false;
  }
}
}
container system-state {           // SID 1716
  container clock {                 // SID 1717
    leaf current-datetime {         // SID 1719
      type yang:date-and-time;
    }
    leaf boot-datetime {            // SID 1718
      type yang:date-and-time;
    }
  }
}
}
```

A.2. server list

Taken over from [RFC7950] section 7.15.3.

```
module example-server-farm {
  yang-version 1.1;
  namespace "urn:example:server-farm";
  prefix "sfarm";

  import ietf-yang-types {
    prefix "yang";
  }

  list server {                                // SID 60000
    key name;
    leaf name {                                // SID 60001
      type string;
    }
    action reset {                             // SID 60002
      input {
        leaf reset-at {                       // SID 60003
          type yang:date-and-time;
          mandatory true;
        }
      }
      output {
        leaf reset-finished-at {              // SID 60004
          type yang:date-and-time;
          mandatory true;
        }
      }
    }
  }
}
```

A.3. interfaces

Excerpt of the YANG module ietf-interfaces [RFC7223].

```
module ietf-interfaces {  
  container interfaces {  
    list interface {  
      key "name";  
      leaf name {  
        type string;  
      }  
      leaf description {  
        type string;  
      }  
      leaf type {  
        type identityref {  
          base interface-type;  
        }  
        mandatory true;  
      }  
  
      leaf enabled {  
        type boolean;  
        default "true";  
      }  
  
      leaf link-up-down-trap-enable {  
        if-feature if-mib;  
        type enumeration {  
          enum enabled {  
            value 1;  
          }  
          enum disabled {  
            value 2;  
          }  
        }  
      }  
    }  
  }  
}
```

A.4. Example-port

Notification example defined within this document.

```

module example-port {
  ...
  notification example-port-fault { // SID 60010
    description
      "Event generated if a hardware fault on a
       line card port is detected";
    leaf port-name { // SID 60011
      type string;
      description "Port name";
    }
    leaf port-fault { // SID 60012
      type string;
      description "Error condition detected";
    }
  }
}

```

A.5. IP-MIB

The YANG translation of the SMI specifying the IP-MIB [RFC4293], extended with example SID numbers, yields:

```

module IP-MIB {
  import IF-MIB {
    prefix if-mib;
  }
  import INET-ADDRESS-MIB {
    prefix inet-address;
  }
  import SNMPv2-TC {
    prefix smiv2;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import yang-smi {
    prefix smi;
  }
  import ietf-yang-types {
    prefix yang;
  }

  container ip { // SID 60020
    list ipNetToPhysicalEntry { // SID 60021
      key "ipNetToPhysicalIfIndex ipNetToPhysicalNetAddressType ipNetToPhysicalNetAddress";
      leaf ipNetToPhysicalIfIndex { // SID 60022
        type if-mib:InterfaceIndex;
      }
    }
  }
}

```



```
    }
    leaf ipNetToPhysicalNetAddressType { // SID 60023
        type inet-address:InetAddressType;
    }
    leaf ipNetToPhysicalNetAddress { // SID 60024
        type inet-address:InetAddress;
    }
    leaf ipNetToPhysicalPhysAddress { // SID 60025
        type yang:phys-address {
            length "0..65535";
        }
    }
    leaf ipNetToPhysicalLastUpdated { // SID 60026
        type yang:timestamp;
    }
    leaf ipNetToPhysicalType { // SID 60027
        type enumeration {
            enum "other" {
                value 1;
            }
            enum "invalid" {
                value 2;
            }
            enum "dynamic" {
                value 3;
            }
            enum "static" {
                value 4;
            }
            enum "local" {
                value 5;
            }
        }
    }
    leaf ipNetToPhysicalState { // SID 60028
        type enumeration {
            enum "reachable" {
                value 1;
            }
            enum "stale" {
                value 2;
            }
            enum "delay" {
                value 3;
            }
            enum "probe" {
                value 4;
            }
        }
    }
}
```

```
        enum "invalid" {
            value 5;
        }
        enum "unknown" {
            value 6;
        }
        enum "incomplete" {
            value 7;
        }
    }
}
leaf ipNetToPhysicalRowStatus {          // SID 60029
    type smiv2:RowStatus;
} // list ipNetToPhysicalEntry
} // container ip
} // module IP-MIB
```

Appendix B. Comparison with LWM2M

B.1. Introduction

CoMI and LWM2M [OMA], both, provide RESTful device management services over CoAP. Differences between the designs are highlighted in this section.

The intent of the LWM2M protocol is to provide a single protocol to control and manage IoT devices. This means the IoT device implements and uses the same LWM2M agent function for the actuation and sensing features of the IoT device as well as for the management of the IoT device. The intent of CoMI Interface as described in the Abstract section of this document is to provide management of constrained devices and devices in constrained networks using RESTCONF and YANG. This implies that the device, although reusing the CoAP protocol, would need a separate CoAP based agent in the future to control the actuation and sensing features of the device and another CoMI agent that performs the management functions.

It should be noted that the mapping of a LWM2M server to YANG is specified in [YANGlwm2m]. The converted server can be invoked with CoMI as specified in this document.

For the purposes of managing IoT devices the following points related to the protocols compare how management resources are defined, identified, encoded and updated.

B.2. Defining Management Resources

Management resources in LWM2M (LWM2M objects) are defined using a standardized number. When a new management resource is defined, either by a standards organization or a private enterprise, the management resource is registered with the Open Mobile Naming Authority [OMNA] in order to ensure different resource definitions do not use the same identifier. CoMI, by virtue of using YANG as its data modeling language, allows enterprises and standards organizations to define new management resources (YANG nodes) within YANG modules without having to register each individual management resource. Instead YANG modules are scoped within a registered name space. As such, the CoMI approach provides additional flexibility in defining management resources. Likewise, since CoMI utilizes YANG, existing YANG modules can be reused. The flexibility and reuse capabilities afforded to CoMI can be useful in management of devices like routers and switches in constrained networks. However for management of IoT devices, the usefulness of this flexibility and applicability of reuse of existing YANG modules may not be warranted. The reason is that IoT devices typically do not require complex sets of configuration or monitoring operations required by devices like a router or a switch. To date, OMA has defined approximately 15 management resources for constrained and non-constrained mobile or fixed IoT devices while other 3rd Party SDOs have defined another 10 management resources for their use in non-constrained IoT devices. Likewise, the Constrained Object Language [I-D.veillette-core-cool] which is used by CoMI when managing constrained IoT devices uses YANG schema item identifiers, which are registered with IANA, in order to define management resources that are encoded using CBOR when targeting constrained IoT Devices.

B.3. Identifying Management Resources

As LWM2M and CoMI can similarly be used to manage IoT devices, comparison of the CoAP URIs used to identify resources is relevant as the size of the resource URI becomes applicable for IoT devices in constrained networks. LWM2M uses a flat identifier structure to identify management resources and are identified using the LWM2M object's identifier, instance identifier and optionally resource identifier (for access to and object's attributes). For example, identifier of a device object (object id = 3) would be "/3/0" and identification of the device object's manufacturer attribute would be "/3/0/0". Effectively LWM2M identifiers for management resources are between 4 and 10 bytes in length.

CoMI is expected to be used to manage constrained IoT devices. CoMI utilizes the YANG schema item identifier [SID] that identify the resources. CoMI recommends that IoT device expose resources to

identify the data stores and event streams of the CoMI agent. Individual resources (e.g., device object) are not directly identified but are encoded within the payload. As such the identifier of the CoMI resource is smaller (4 to 7 bytes) but the overall payload size isn't smaller as resource identifiers are encoded on the payload.

B.4. Encoding of Management Resources

LWM2M provides a separation of the definition of the management resources from how the payloads are encoded. As of the writing of this document LWM2M encodes payload data in Type-length-value (TLV), JSON or plain text formats. JSON encoding is the most common encoding scheme with TLV encoding used on the simplest IoT devices. CoMI's use of CBOR provides a more efficient transfer mechanism [RFC7049] than the current LWM2M encoding formats.

In situations where resources need to be modified, CoMI uses the CoAP PATCH operation resources only require a partial update. LWM2M does not currently use the CoAP PATCH operation but instead uses the CoAP PUT and POST operations which are less efficient.

Authors' Addresses

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com

Michel Veillette
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

core
Internet-Draft
Intended status: Informational
Expires: September 22, 2016

P. van der Stok
Consultant
C. Bormann
Universitaet Bremen TZI
A. Sehgal
Consultant
March 21, 2016

Patch and Fetch Methods for Constrained Application Protocol (CoAP)
draft-vanderstok-core-etch-00

Abstract

The existing Constrained Application Protocol (CoAP) methods only allow access to a complete resource. This does not permit applications to access parts of a resource. In case of resources with larger or complex data, or in situations where a resource continuity is required, replacing or requesting the whole resource is undesirable. Several applications using CoAP will need to perform partial resource accesses.

Similar to HTTP, the existing Constrained Application Protocol (CoAP) GET method only allows the specification of a URI and request parameters in CoAP options, not the transfer of a request payload detailing the request. This leads to some applications to using POST where actually a cacheable, idempotent, safe request is desired.

Again similar to HTTP, the existing Constrained Application Protocol (CoAP) PUT method only allows to replace a complete resource. This also leads applications to use POST where actually a cacheable, possibly idempotent request is desired.

This specification adds new CoAP methods, FETCH, to perform the equivalent of a GET with a request body; and the twin methods PATCH and iPATCH, to modify parts of an existing CoAP resource.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. FETCH	3
1.2. PATCH and iPATCH	4
1.3. Requirements Language	4
1.4. Terminology and Acronyms	4
2. FETCH Method	4
2.1. The Content-Format Option	6
2.2. Working with Observe	6
2.3. Working with Block	6
2.4. FETCH discussion	6
3. PATCH and iPATCH Methods	6
3.1. Simple Examples for PATCH and iPATCH	8
3.2. Response Codes	10
3.3. Option Numbers	10
3.4. Error Handling	10
4. Discussion	12
5. Security Considerations	12
6. IANA Considerations	13
7. Acknowledgements	14
8. Change log	14
9. References	14
9.1. Normative References	14
9.2. Informative References	15
Authors' Addresses	16

1. Introduction

This specification defines the new Constrained Application Protocol (CoAP) [RFC7252] methods, `FETCH`, `PATCH` and `iPATCH`, which are used to access and update parts of a resource.

1.1. `FETCH`

The CoAP `GET` method [RFC7252] is used to obtain the representation of a resource, where the resource is specified by a URI and additional request parameters can additionally shape the representation. This has been modelled after the HTTP `GET` operation and the REST model in general.

In HTTP, a resource is often used to search for information, and existing systems varyingly use the HTTP `GET` and `POST` methods to perform a search. Often a `POST` method is used for the sole reason that a larger set of parameters to the search can be supplied in the request body than can comfortably be transferred in the URI with a `GET` request. The draft [I-D.snell-search-method] proposes a `SEARCH` method that is similar to `GET` in most properties but enables sending a request body as with `POST`. The `FETCH` method defined in the present specification is inspired by [I-D.snell-search-method], which updates the definition and semantics of the HTTP `SEARCH` request method previously defined by [RFC5323]. However, there is no intention to limit `FETCH` to search-type operations, and the resulting properties may not be the same as those of HTTP `SEARCH`.

A major problem with `GET` is that the information that controls the request needs to be bundled up in some unspecified way into the URI. Using the request body for this information has a number of advantages:

- o The client can specify a media type (and a content encoding), enabling the server to unambiguously interpret the request parameters in the context of that media type. Also, the request body is not limited by the character set limitations of URIs, enabling a more natural (and more efficient) representation of certain domain-specific parameters.
- o The request parameters are not limited by the maximum size of the URI. In HTTP, that is a problem as the practical limit for this size varies. In CoAP, another problem is that the block-wise transfer is not available for transferring large URI options in multiple rounds.

As an alternative to using `GET`, many implementations make use of the `POST` method to perform extended requests, even if they are

semantically idempotent, safe, and even cacheable, to be able to pass along the input parameters within the request payload as opposed to using the request URI.

The FETCH method provides a solution that spans the gap between the use of GET and POST. As with POST, the input to the FETCH operation is passed along within the payload of the request rather than as part of the request URI. Unlike POST, however the semantics of the FETCH method are more specifically defined.

1.2. PATCH and iPATCH

PATCH is also specified for HTTP in [RFC5789]. Most of the motivation for PATCH described in [RFC5789] also applies here. iPATCH is the idempotent version of PATCH.

The PUT method exists to overwrite a resource with completely new contents, and cannot be used to perform partial changes. When using PUT for partial changes, proxies and caches, and even clients and servers, may get confused as to the result of the operation. PATCH was not adopted in an early design stage of CoAP, however, it has become necessary with the arrival of applications that require partial updates to resources (e.g. [I-D.vanderstok-core-comi]). Using PATCH avoids transferring all data associated with a resource in case of modifications, thereby not burdening the constrained communication medium.

This document relies on knowledge of the PATCH specification for HTTP [RFC5789]. This document provides extracts from [RFC5789] to make independent reading possible.

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.4. Terminology and Acronyms

This document uses terminology defined in [RFC5789] and [RFC7252].

2. FETCH Method

The CoAP FETCH method is used to obtain a representation of a resource, giving a number of request parameters. Unlike the CoAP GET method, which requests that a server return a representation of the resource identified by the effective request URI (as defined by

[RFC7252]), the FETCH method is used by a client to ask the server to produce a representation as described by the request parameters (including the request options and the payload) based on the resource specified by the effective request URI. The payload returned in response to a FETCH cannot be assumed to be a complete representation of the resource identified by the effective request URI.

Together with the request options, the body of the request (which may be constructed from multiple payloads using the block protocol [I-D.ietf-core-block]) defines the request parameters. Implementations MAY use a request body of any content type with the FETCH method; it is outside the scope of this document how information about admissible content types is obtained by the client (although we can hint that form relations ([I-D.hartke-core-apps]) might be a preferred way).

FETCH requests are both safe and idempotent with regards to the resource identified by the request URI. That is, the performance of a fetch is not intended to alter the state of the targeted resource. (However, while processing a search request, a server can be expected to allocate computing and memory resources or even create additional server resources through which the response to the search can be retrieved.)

A successful response to a FETCH request is expected to provide some indication as to the final disposition of the requested operation. If a successful response includes a body payload, the payload is expected to describe the results of the FETCH operation.

Depending on the response code as defined by [RFC7252], the response to a FETCH request is cacheable; the request body is part of the cache key. Specifically, 2.05 "Content" response codes, the responses for which are cacheable, are a usual way to respond to a FETCH request. (Note that this aspect differs markedly from [I-D.snell-search-method].) (Note also that caches that cannot use the request payload as part of the cache key will not be able to cache responses to FETCH requests at all.) The Max-Age option in the response has equivalent semantics to its use in a GET.

The semantics of the FETCH method change to a "conditional FETCH" if the request message includes an If-Match, or If-None-Match option ([RFC7252]). A conditional FETCH requests that the query be performed only under the circumstances described by the conditional option(s). It is important to note, however, that such conditions are evaluated against the state of the target resource itself as opposed to the results of the FETCH operation.

TODO This needs some additional text on what an ETag on a FETCH result means.

2.1. The Content-Format Option

A FETCH request MUST include a Content-Format option to specify the media type and content encoding of the request body.

2.2. Working with Observe

The Observe option [RFC7641] can be used with a FETCH request as it can be used with a GET request.

2.3. Working with Block

The Block1 option [I-D.ietf-core-block] can be used with a FETCH request as it would be used with a POST request; the Block2 option can then be used as with GET or POST.

2.4. FETCH discussion

One property of FETCH that may be non-obvious is that a FETCH request cannot be generated from a link alone, but also needs a way to generate the request payload. Again, form relations ([I-D.hartke-core-apps]) may be able to fill parts of this gap.

3. PATCH and iPATCH Methods

The PATCH and iPATCH methods request that a set of changes described in the request payload is applied to the target resource of the request. The set of changes is represented in a format identified by a media type. If the Request-URI does not point to an existing resource, the server MAY create a new resource with that URI, depending on the patch document type (whether it can logically modify a null resource) and permissions, etc. Creation of a new resource would result in a 2.01 (Created) Response Code dependent of the patch document type.

Restrictions to a PATCH or iPATCH request can be made by including the If-Match or If-None-Match options in the request (see Section 5.10.8.1 and 5.10.8.2 of [RFC7252]). If the resource could not be created or modified, then an appropriate Error Response Code SHOULD be sent.

The difference between the PUT and PATCH requests is extensively documented in [RFC5789].

The PATCH method is not safe and not idempotent, as with the HTTP PATCH method specified in [RFC5789].

The iPATCH method is not safe but idempotent, as with the CoAP PUT method specified in [RFC7252], Section 5.8.3.

A client can mark a request as idempotent by using the iPATCH method instead of the PATCH method. This is the only difference between the two. The indication of idempotence may enable the server to keep less state about the interaction; some constrained servers may only implement the iPATCH variant for this reason.

PATCH and iPATCH are both atomic. The server MUST apply the entire set of changes atomically and never provide a partially modified representation to a concurrently executed GET request. Given the constrained nature of the servers, most servers will only execute CoAP requests consecutively, thus preventing a concurrent partial overlapping of request modifications. Resuming, modifications MUST NOT be applied to the server state when an error occurs or only a partial execution is possible on the resources present in the server.

The atomicity applies to a single server. When a PATCH or iPATCH request is multicast to a set of servers, each server can either execute all required modifications or not. It is not required that all servers execute all modifications or none. An Atomic Commit protocol that provides multiple server atomicity is out of scope.

A PATCH or iPATCH response can invalidate a cache as with the PUT response. Caching behaviour as function of the successful (2.xx) response codes for PATCH or iPATCH are:

- o A 2.01 (Created) response invalidates any cache entry for the resource indicated by the Location-* Options; the payload is a representation of the action result.
- o A 2.04 (Changed) response invalidates any cache entry for the target resource; the payload is a representation of the action result.

There is no guarantee that a resource can be modified with PATCH or iPATCH. Servers MUST ensure that a received PATCH body is appropriate for the type of resource identified by the target resource of the request.

When a request is intended to effect a partial update of a given resource, clients cannot use PUT while supplying just the update, but are free to use PATCH or iPATCH.

3.1. Simple Examples for PATCH and iPATCH

The example is taken over from [RFC6902], which specifies a JSON notation for PATCH operations. A resource located at `coap://www.example.com/object` contains a target JSON document.

JSON document original state:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

REQ: iPATCH CoAP://www.example.com/object
Content-Format: application/json-patch+json

```
[
  { "op": "replace", "path": "x-coord", "value": 45 }
]
```

RET: CoAP 2.04 Changed

JSON document final state:

```
{
  "x-coord": 45,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

This example illustrates use of an idempotent modification to the `x-coord` member of the existing resource `"object"`. The 2.04 (Changed) response code is conform with the CoAP PUT method.

The same example using the Content-Format `application/merge-patch+json` from [RFC7396] looks like:

JSON document original state:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

```
REQ: iPATCH CoAP://www.example.com/object
Content-Format: 52 (application/merge-patch+json)
{ "x-coord":45}
```

RET: CoAP 2.04 Changed

JSON document final state:

```
{
  "x-coord": 45,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

The examples show the use of the iPATCH method, but the use of the PATCH method would have led to the same result. Below a non-idempotent modification is shown. Because the action is non-idempotent, iPATCH returns an error, while PATCH executes the action.

JSON document original state:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

REQ: iPATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)

```
[
  { "op": "add", "path": "foo/1", "value": "bar" }
]
```

RET: CoAP 4.12 Precondition Failed

JSON document final state is unchanged

REQ: PATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)

```
[
  { "op": "add", "path": "foo/1", "value": "bar" }
]
```

RET: CoAP 2.04 Changed

JSON document final state:

```
{
  "x-coord": 45,
  "y-coord": 45,
  "foo": ["bar", "bar", "baz"]
}
```

3.2. Response Codes

PATCH and iPATCH for CoAP adopt the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252].

3.3. Option Numbers

PATCH and iPATCH for CoAP adopt the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

3.4. Error Handling

A PATCH or iPATCH request may fail under certain known conditions. These situations should be dealt with as expressed below.

Malformed PATCH or iPATCH payload: If a server determines that the payload provided with a PATCH or iPATCH request is not properly formatted, it can return a 4.00 (Bad Request) CoAP error. The

definition of a malformed payload depends upon the CoAP Content-Format specified with the request.

Unsupported PATCH or iPATCH payload: In case a client sends payload that is inappropriate for the resource identified by the Request-URI, the server can return a 4.15 (Unsupported Content-Format) CoAP error. The server can determine if the payload is supported by checking the CoAP Content-Format specified with the request.

Unprocessable request: This situation occurs when the payload of a PATCH request is determined as valid, i.e. well-formed and supported, however, the server is unable to or incapable of processing the request. The server can return a 4.22 (Unprocessable Entity) CoAP error. More specific scenarios might include situations when:

- * the server has insufficient computing resources to complete the request successfully -- 4.13 (Request Entity Too Large) CoAP Response Code (see below),
- * the resource specified in the request becomes invalid by applying the payload -- 4.09 (Conflict) CoAP Response Code (see below)),

In case there are more specific errors that provide more insight into the problem, then those should be used.

Resource not found: The 4.04 (Not Found) error should be returned in case the payload of a PATCH request cannot be applied to a non-existent resource.

Failed precondition: In case the client uses the conditional If-Match or If-None-Match option to define a precondition for the PATCH request, and that precondition fails, then the server can return the 4.12 (Precondition Failed) CoAP error.

Request too large: If the payload of the PATCH request is larger than a CoAP server can process, then it can return the 4.13 (Request Entity Too Large) CoAP error.

Conflicting state: If the modification specified by a PATCH or iPATCH request causes the resource to enter an inconsistent state that the server cannot resolve, the server can return the 4.09 (Conflict) CoAP response. The server SHOULD generate a payload that includes enough information for a user to recognize the source of the conflict. The server MAY return the actual resource state to provide the client with the means to create a new consistent resource state. Such a situation might be encountered

when a structural modification is applied to a configuration data-store, but the structures being modified do not exist.

Concurrent modification: Resource constrained devices might need to process requests in the order they are received. In case requests are received concurrently to modify the same resource but they cannot be queued, the server can return a 5.03 (Service unavailable) CoAP response code.

Conflict handling failure: If the modification implies the reservation of resources or the waiting on conditions to become true, leading to a too long request execution time, the server can return 5.03 (service unavailable) response code.

It is possible that other error situations, not mentioned here, are encountered by a CoAP server while processing the PATCH request. In these situations other appropriate CoAP status codes can also be returned.

4. Discussion

Adding three new methods to CoAP's existing four may seem like a major change. However, both FETCH and the two PATCH variants fit well into the REST paradigm and have been anticipated on the HTTP side. Adding both a non-idempotent and an idempotent PATCH variant allows to keep interoperability with HTTP's PATCH method as well as the use/indication of an idempotent PATCH if that is possible, saving significant effort on the server side.

Interestingly, the three new methods fit into the old table of methods with a surprising similarity in the idempotence and safety attributes:

Code	Name	Code	Name	safe	idempotent
0.01	GET	0.05	FETCH	yes	yes
0.02	POST	0.06	PATCH	no	no
0.03	PUT	0.07	iPATCH	no	yes
0.04	DELETE			no	yes

5. Security Considerations

This section analyses the possible threats to the CoAP FETCH and PATCH or iPATCH methods. It is meant to inform protocol and application developers about the security limitations of CoAP FETCH and PATCH or iPATCH as described in this document.

The FETCH method is subject to the same general security considerations as all CoAP methods as described in [RFC7252].

The security consideration of section 15 of [RFC2616], section 11 of [RFC7252], and section 5 of [RFC5789] also apply.

The security considerations for PATCH or iPATCH are nearly identical to the security considerations for PUT ([RFC7252]). The mechanisms used for PUT can be used for PATCH or iPATCH as well.

PATCH or iPATCH are secured following the CoAP recommendations as specified in section 9 of [RFC7252]. When additional security techniques are standardized for CoAP, PATCH or iPATCH can also be (and need to be) secured by those new techniques.

6. IANA Considerations

IANA is requested to add the following entries to the sub-registry "CoAP Method Codes":

Code	Name	Reference
0.05	FETCH	[RFCthis]
0.06	PATCH	[RFCthis]
0.07	iPATCH	[RFCthis]

The FETCH method is idempotent and safe, and it returns the same response codes that GET can return, plus 4.15 "Unsupported Content-Format" with the same semantics as with POST.

The PATCH method is neither idempotent nor safe. It returns the same response codes that POST can return, plus 4.09 "Conflict" with the semantics specified in Section 3.4.

IANA is requested to add the following code to the sub-registry "CoAP response codes":

Code	Name	Reference
4.09	Conflict	[RFCthis]

IANA is requested to add entries to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry:

Media Type	Encoding	ID	Reference
application/json-patch+json		51	[RFC6902]
application/merge-patch+json		52	[RFC7396]

7. Acknowledgements

Klaus Hartke has pointed out some essential differences between CoAP and HTTP concerning PATCH, and found a number of problems in an earlier version of Section 2. We are grateful for discussions with Christian Amsuss, Timothy Carey, Paul Duffy, Kovatsch Matthias, Michel Veillette, Michael Verschoor, Thomas Watteyne, and Gengyu Wei.

8. Change log

When published as a RFC, this section needs to be removed.

Version 00 is a composition from draft-vanderstok-core-patch-03 and draft-bormann-core-coap-fetch-00 and replaces these two drafts.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <<http://www.rfc-editor.org/info/rfc2616>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <<http://www.rfc-editor.org/info/rfc5789>>.

- [RFC6902] Bryan, P., Ed. and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902, April 2013, <<http://www.rfc-editor.org/info/rfc6902>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <<http://www.rfc-editor.org/info/rfc7396>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

9.2. Informative References

- [RFC5323] Reschke, J., Ed., Reddy, S., Davis, J., and A. Babich, "Web Distributed Authoring and Versioning (WebDAV) SEARCH", RFC 5323, DOI 10.17487/RFC5323, November 2008, <<http://www.rfc-editor.org/info/rfc5323>>.
- [I-D.vanderstok-core-comi]
Stok, P. and A. Bierman, "CoAP Management Interface", draft-vanderstok-core-comi-09 (work in progress), March 2016.
- [I-D.hartke-core-apps]
Hartke, K., "CoRE Application Descriptions", draft-hartke-core-apps-03 (work in progress), February 2016.
- [I-D.snell-search-method]
Reschke, J., Malhotra, A., and J. Snell, "HTTP SEARCH Method", draft-snell-search-method-00 (work in progress), April 2015.

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Anuj Sehgal
Consultant

Email: anuj@iurs.org

Internet Engineering Task Force
INTERNET-DRAFT
Intended Status: Standard Track
Expires: April 19, 2016

Vasu K
Rahul A J
Yangneng
Huawei
Oct 19, 2015

Service Provisioning for Constrained Devices
draft-vasu-core-ace-service-provisioning-00

Abstract

As more constrained devices are integrating with current Internet, the ubiquitous computing in scenarios like smart home is very important. In smart home, the constrained devices (ex. thermostat) need to be provisioned in such a way that it can inter-operate with any kind of devices like other constrained devices (ex. Air conditioner) or client devices (ex. smart phone). This document provides a method to support service provisioning based on pre-configured admission and resource control policies, where this method explains device's service access in two different use cases: first provisioning the service when a constrained device accessing the service provided by other constrained device, second, accessing the service provided by constrained device from the client device (non constrained device).

Status of this Memo This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on April 19, 2016.

Copyright and License Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1	Introduction	3
2	Motivation	5
3	Terminology	5
4	System Architecture	6
5	Network Topology	9
6	Operations	10
6.1	Register Service	10
6.3.1	Resource Control	14
6.4	Search for services by device	18
6.5	Service request and response	18
7	Security Considerations	22
8	IANA Considerations	22
9	References	23
9.1	Normative References	23
9.2	Informative References	23
10	Acknowledgements	24
	Authors' Addresses	24

1 Introduction

The work on Constrained Restful Environment (CoRE) aimed to realize the restful architecture for constrained devices [RFC7228] in constrained networks [RFC4944]. The CoRE work group has recently standardized constrained application protocol (CoAP) [RFC7252] for interacting with constrained resources where general HTTP is not memory/energy efficient. The use of web linking for resources description and discovery hosted by constrained web servers is specified by CoRE [RFC6690]. Even though, CoAP allows the direct resource access for constrained devices, it is not advisable for direct access of resources in networks where multicast procedures are infeasible due to heavy network load, and the networks where sleepy nodes exist. So, the CoRE working group comes up with a solution called resource directory (RD) [draft-ietf-core-resource-directory] to host the devices service information, and allow other devices to perform lookup procedures through .well-known/core path to resources.

The services advertised by these constrained devices need to be commissioned and provisioned properly to allow other devices to access it. CoRE RD solution is a directory based solution that depends on CoAP protocol. CoRE RD solution uses registration/update/delete/lookup procedures for service registration, service update, deleting service, lookup of services respectively. Service commissioning is a method which verifies a pre-registered services with special commissioning tools/agents. These tools can be tablets or special embedded devices which initially store the devices identifications in secure manner. Once the services are advertised by any device, those services need to be verified using commissioner. CoRE RD provides a standard procedure to interact with commissioner, where commissioner acts like a client device to look up and verify the advertised services. Once the commissioner verifies the pre-registered services, commissioner can put some policy rules on services hosted by devices for resource control. These rules defined on (1) how to access the services either with other constrained devices or client devices, and (2) on operational instructions.

Architecture is defined to authenticate and authorize client requests for a resource on a server using logical entities such as client(C), client authorization manager(CAM), server(S), and server authorization manager(SAM) [draft-gerdes-ace-actors]. The main goal of delegated CoAP authentication and authorization framework (DCAF) is the setup of a datagram transport layer security channel between two nodes to securely transmit authorization tickets [draft-gerdes-core-dcaf-authorize]. The CAM sends an access request message on behalf of client by embedding requested permissions in client authorization information (CAI) field of access request message to

SAM. A ticket grant message is sent from SAM by embedding the permissions given from the server on a specific resource in server authorization information (SAI) field of ticket grant message to the client. These SAI, CAI use authorization information format (AIF) that describes the permissions requested from access request in a ticket request, where the underlying access control model will be that of an access matrix, which gives a set of permissions for each possible combination of a subject and an object [draft-bormann-core-ace-aif]. This simple information model also doesn't allow conditional access (e.g., "resource /s/tempC is accessible only if client belongs to group1 and does not belong to group2"). Finally, the model does not provide any dynamic functions such as enabling special access for a set of resources that are specific to a subject. But, the services provided by resources in constrained environment, need to be authorized and controlled conditionally based on some service level agreements or preconfigured policies on resource control.

Considering an example use case scenario such as thermostat device measures the current room temperature, and can service for air conditioner device to set automatic temperatures. In a smart home, user wants to regulate his room temperature automatically using his airconditioner device. Here, this airconditioner device can adjust its temperature to either cool the room or heat the room by accessing the service provided by the thermostat. Suppose this user leaves the home in the morning in hot summer and leaves the office in the evening to reach to home. But, before he reaches his room he wants to make his room cool enough. So he has to switch on the airconditioner from his mobile one hour before he leaves the office. So, before adjusting his airconditioner to make the room cool enough, he might have to know the current room temperature. Thus he access the service provided by the thermostat to read the room temperature and adjust the airconditioner. However, there is a problem here on how to access these services which are provided by user's home devices itself, what is the authenticity level to access from outside the home, even within home what is the access control/resource control of these devices because the neighboring device which are not authenticated can also access these service if those devices are within the constrained network range. Finally it is important to admit access of the service by client based on the configuration policies so that the devices can be protected from hazardous conditions, and allows only pre-agreed operations on devices.

The service provisioning presented in this document provides a method to support admission, and resource control policies using commissioning procedure. The method explains the device's service access in two different use cases: first provisioning the service when a constrained device accessing the service provided by other

constrained device, second, accessing the service provided by constrained device from the client device. Even though it is out of scope of the present document, it also considers a secure way of service commissioning as part of security.

2 Motivation

CORE RD solution provides various automated operations such as service registrations, service update, service removal, and service lookups initiated by endpoints and clients. However, managing this centralized directory server by allowing authorized users to perform these tasks, setting some service level agreements on clients to access these services, and providing limited or scope oriented lookups by other endpoints or clients require efficient service provisioning mechanism. The service provisioning method presented in this document deals on how a registered service from devices can be accessed by various clients or other devices. Moreover, it also provides a method for handling this resource/service access control mechanism using web service model for efficient service provisioning from outside the constrained home environment.

3 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

- o "CORE", CORE is a Constrained RESTful Environment providing a framework for resource-oriented application intended to run on constrained networks [RFC7228].
- o "COAP" The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and networks [RFC7252].
- o "RD" The Resource Directory (RD) is a directory based server to host the descriptions of resources and allowing the lookups to be performed for those resources by various client devices.
- o "Commissioner" Commissioning agent is tool/device that verifies the devices operation, integrity check with the network.
- o "Constrained Device" These are embedded computing devices that are expected to be as resource constrained in terms of RAM/ROM size, and to be deployed with the constrained environment such as 6LoWPAN Networks.

- o "Client" A client device is like resource constrained client such as other constrained device (ex. Air conditioner) or rich client devices such as Mobile/Laptop/Tablet etc, which access the services hosted by constrained devices (ex. thermostat).
- o "Provisioning Server" this server is a process of verifying service requester, providing access controls or admission controls on resources to be accessed and inter-operating with various devices without bothering about kind of network protocols used. It also provides web access model outside the constrained environment.
- o "Device Profile" A device profile comprises a set of attributes that are associated with a particular device. These include services, features, names, descriptions etc.

4 System Architecture

The system architecture is better explained with two different scenarios: (1) Constrained device access the service advertised by other constrained device is as shown in Fig 1. Here, one constrained device such as air-conditioner can access the service such as current room temperature advertised by other constrained device (ex. thermostat). This advertised service is to be commissioned by commissioner, and then it should be set with some admission and resource control policies by provisioning server. And, finally the service is allowed to advertise its service access from other constrained devices. Any device that is interested in that advertised service, need to do service lookup from RD Server. Once obtaining the path to the advertised service, the constrained client device can request a service to the device which hosts the service. Before sending the request, it MUST establish a secure channel between these two nodes [draft-schmitt-ace-twowayauth-for-iot]. Once the incoming request comes from the constrained client device, the device which hosts the service MUST authorize and provision for conditional access of its service from the provisioning server. The notification regarding the registered services to the commissioning agent can be sent from the RD server, which can be implementation specific and left for the user to choose any standard procedures and is out of scope of present document. Detailed operational procedure will be explained in the later sections of this document.

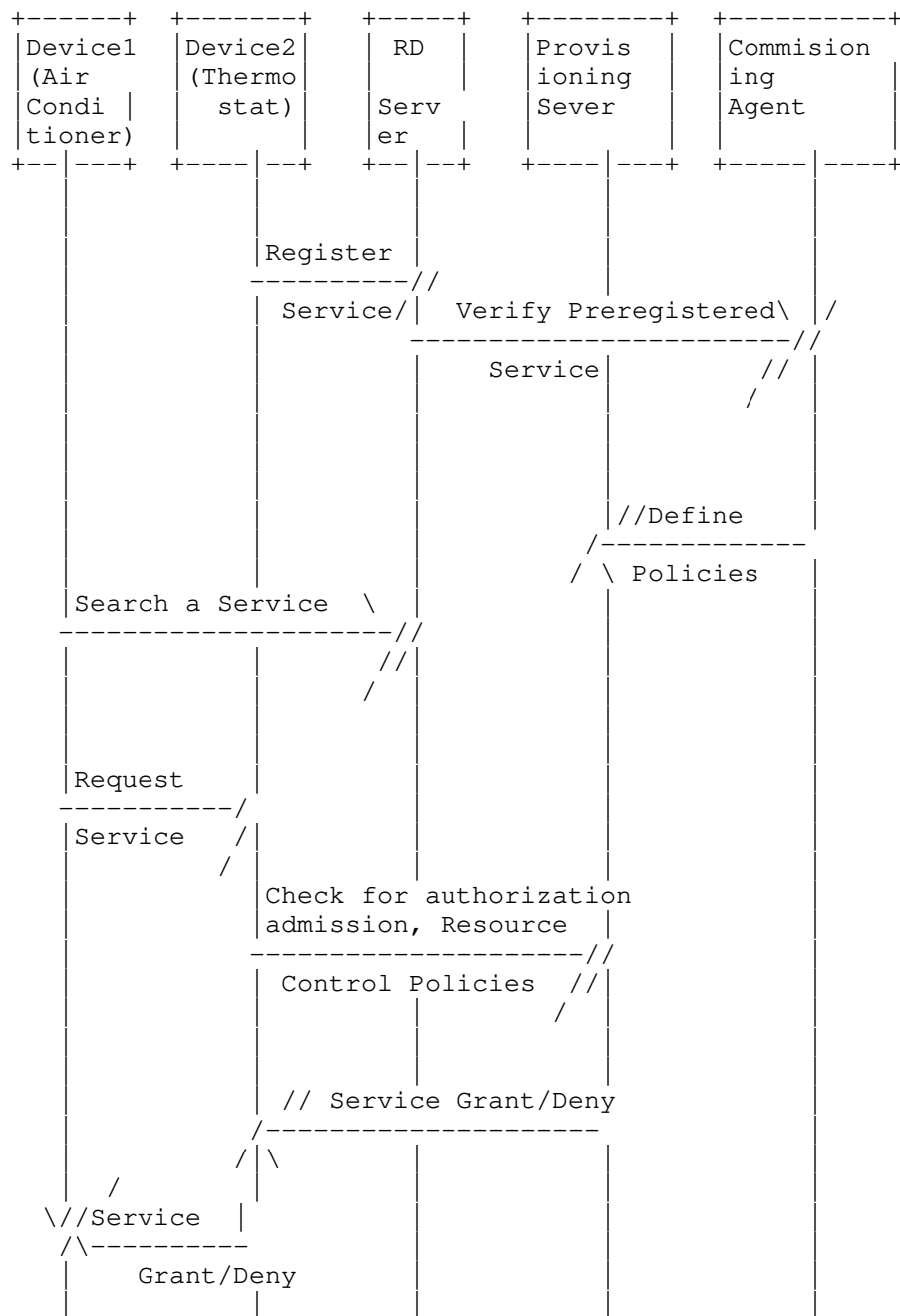


Fig 1. Constrained device accessing service from constrained device

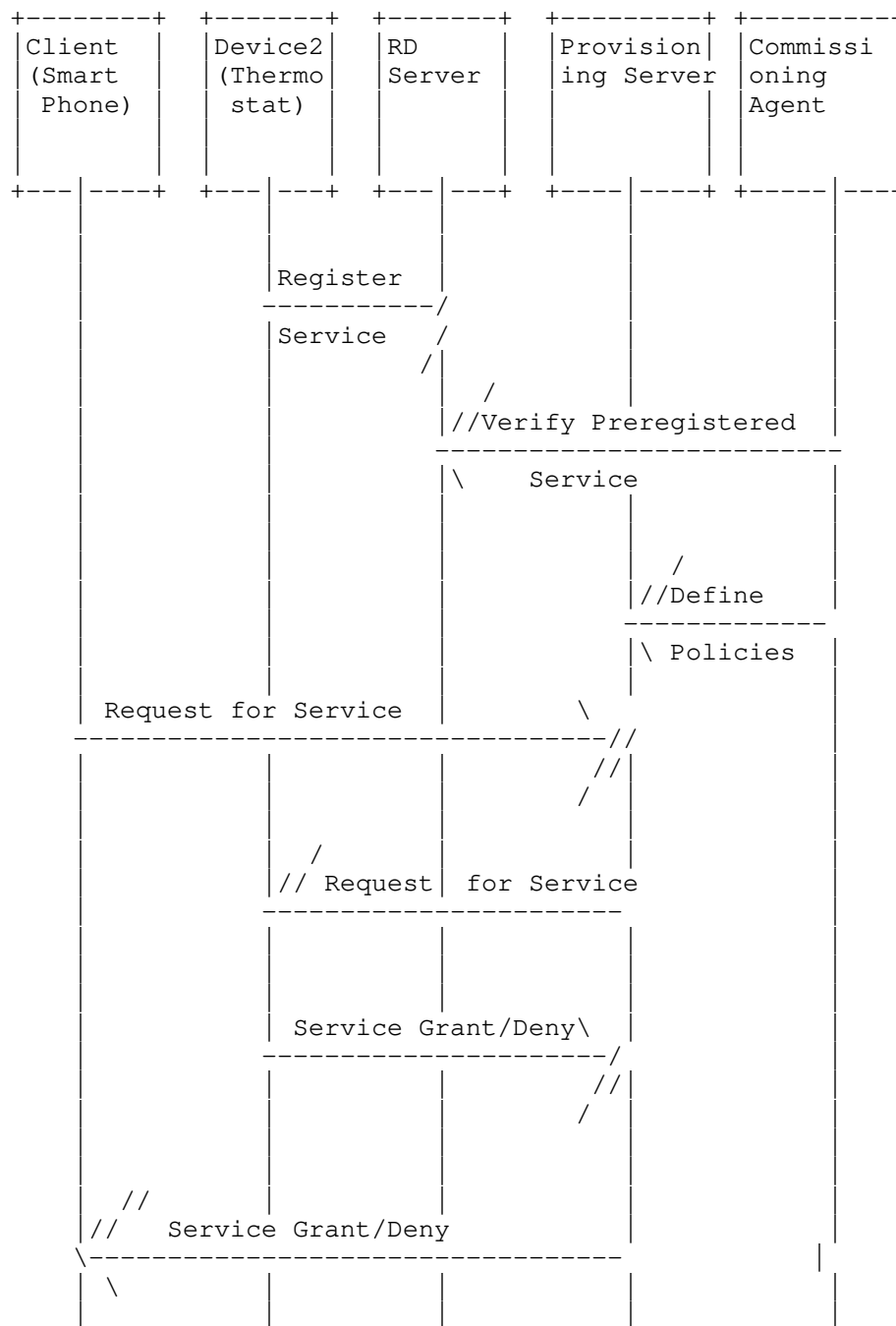


Fig 2. Client accessing service from Constrained device

2) Client device access the service advertised by constrained device is as shown in Fig 2. For example, the client device such as smart phone can access the service (ex. room temperature) advertised by other constrained device (ex. thermostat). The client can access the service within a home environment or outside the home environment. So, in this scenario, the provisioning server maintains the service as a web service.

This advertised service is to be commissioned by commissioner, then to be set with some admission and resource control policies by provisioning server. And, finally the service is allowed to advertise its access from the client devices. Any client that wishes to access this web service looks for corresponding operations provided from the provisioning server.

5 Network Topology

The constrained devices such as Thermostat, Airconditioner may use small memory constrained sensors/actuators for simple services such as cooling/heating the room or just to measure the current room temperature. These memory constrained embedded devices may implement the 6LoWPAN stack such as uIP (provided by Contiki), and provide access for communication to other external queries from client devices such as smart phone which typically implements rich stack TCP/IP. Even though RD server or Provisioning server are shown as separate servers in the LAN as given in Fig 3, these can be hosted on a single server running two different processes. Moreover, the commissioner implements a standard procedure to interact with devices as a separate agent process which is out of scope of the present document and has been left to user's choice while satisfying the mentioned operations in the current draft. On the other hand, these specific operations can be implemented separately as a third party and to be used at the commissioning agent. The lower level communication technology can be implemented either through Bluetooth (BT) or near field communication (NFC) to verify the devices unique ID (for ex. using MAC). Even though, the implementation procedure for commissioner is out of scope for the present document, it is shown as sample interaction with RD server/provisioning server as part of commissioning procedure in subsequent sections. Even though the present document discusses about 6LoWPAN based sensor network, it can be easily moved to any other technology such as Zigbee/BLE/Wireless HART without any changes in the architecture or design, because the present document abstracted the communication networks with their edge routers. The communication and routing mechanisms or procedure between edge router and sensor devices/client devices are out of scope of the present document.

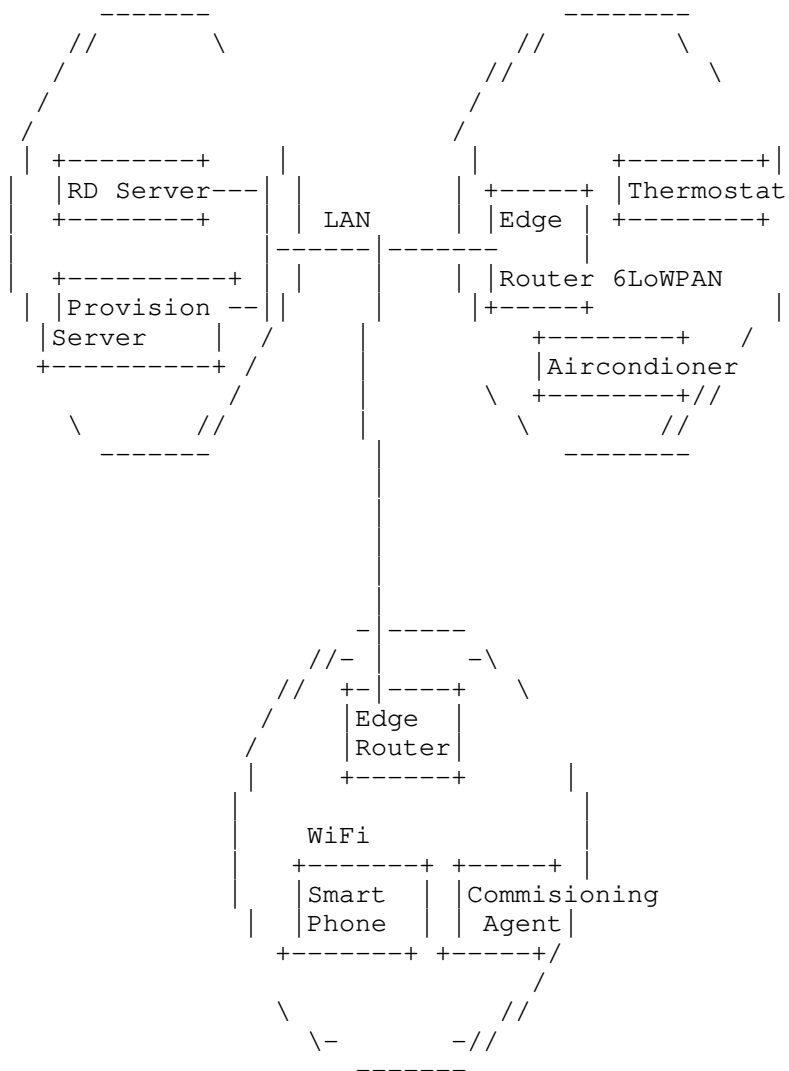


Fig 3. Network Topology

6 Operations

6.1 Register Service

The constrained device which hosts the service MUST register its service with the RD server using its unique identifier (for ex. MAC id, UDDI registry etc.) and IP address as shown in Fig 4. The device MUST send a POST request for registering its service.

Before sending a request, it MUST establish a secure channel between these two nodes [draft-schmitt-ace-twowayauth-for-iot]. Once the service has been registered with the RD server, the RD server may notify the registered information of a device (for ex. its unique identifier and device name) to a commissioning agent.

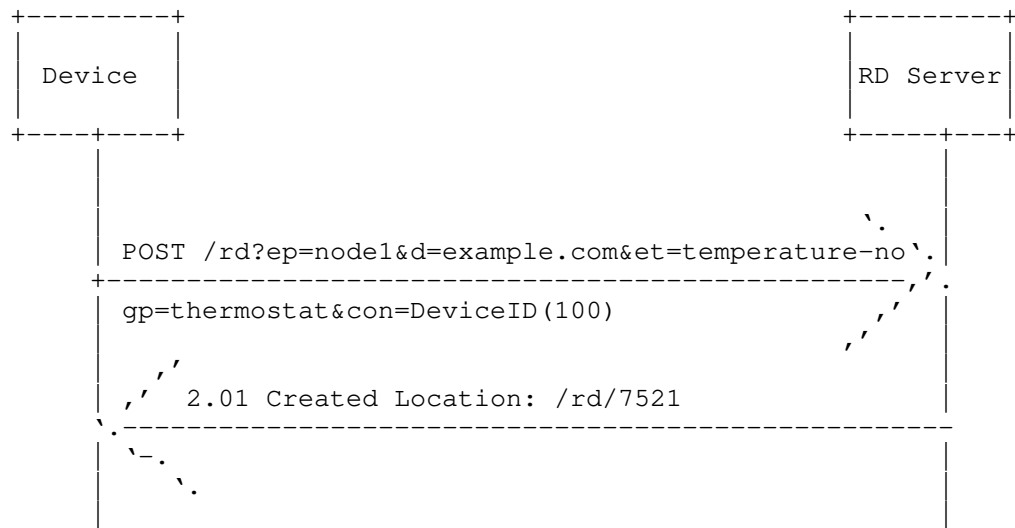


Fig. 4 Registering a Service

6.2 Verify pre-registered service

The commissioning agent MUST verify any pre registered service with the RD server as shown in Fig 5. The commissioning agent sends a GET request for domain lookup. Before sending the request, it MUST establish a secure channel between these two nodes [DTLS][TLS]. Once obtaining the specific domain, it MUST look for the group to which the service belongs. Once obtaining the specific domain and group, it MUST send a service look up with the RD server for the registered service. Once obtaining the service information about a specific device, the commissioning agent MUST verify the registered service. This service information is later used to create service registry in the provisioning server as explained in the following section. The example service information (denoted as SRV) looks like as shown in Fig 6.

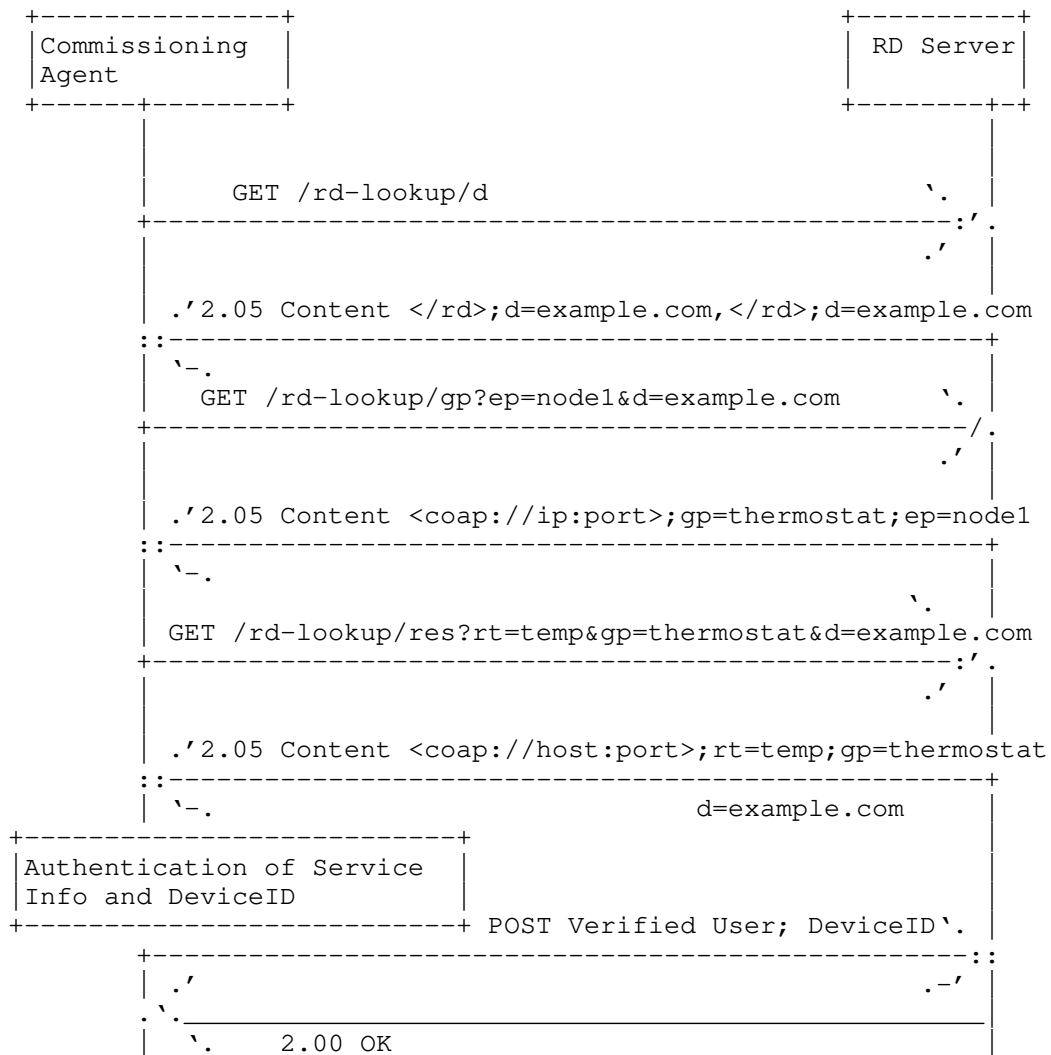


Fig. 5 Verify pre registered service

```

SRV {
    Name: Node1
    Group: Thermostat
    Domain: myhome.com
    Type: Temperature node
    Device ID: 1001
    Device IP: <host:port>
}
    
```

Fig 6. Example Service Informaion

6.3 Define policies on resource control

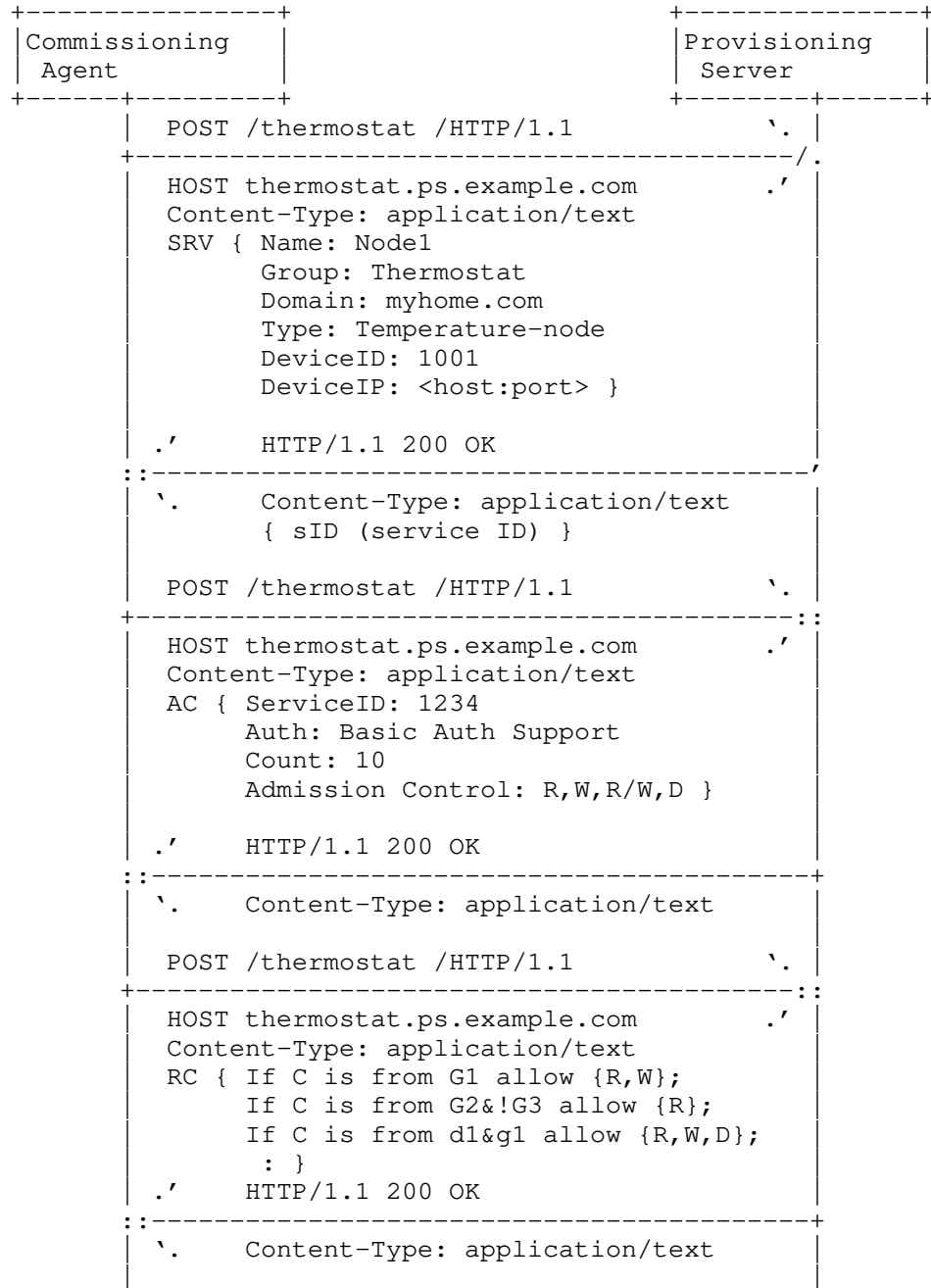


Fig. 7 Defining Policies on Resource and Access Control

Once the hosted service has been verified by commissioning agent (CA), the CA MUST create a service registry with the provisioning server as explained in Fig 7. The provisioning server SHOULD send a service ID as a response back to the commissioning agent after creating the service entry.

This service ID can be later used by the commissioning agent to permanently DELETE the service entry (if required). The commissioning agent MUST create some admission control policies such as read (R), write (W), read/write (R/W), delete (D), number of simultaneous connection on resource etc. on the registered service. Once the admission control policies has been set on a specific device, the resource control policies such as conditional access of a service, quality of service agreements (based on the priority levels set for clients) can be set on that registered service. These conditional access on service can be implemented with simple conditional statements as explained in section 6.3.1 (for ex. "client (c) can access service with only read (R), write (W) permissions if it only belongs to group (g)"). The implementation or information format details of these conditional statements is out of scope of the present document (TBD). The example admission control and resource control policies are as shown in Fig 8, and Fig 9 respectively.

```
AC {
    Service ID: 12345
    Auth: Basic Auth Support
    Count: 10
    Admission Control: R, W, R/W, D
    :
    :
}
```

Fig 8. Example Admission Control Policies

```
RC {
    If c is from g1 allow {R,W}
    If C is from g2 & !g3 {R}
    If C is from d1 & g1 allow {R, W, D}
    :
    :
}
```

Fig 9. Example Resource Control Policies

6.3.1 Resource Control

Resource control policies for constrained devices are expressed in

terms of conditional expressions as explained in Fig. 9. Consider a scenario where we define the client (C) (who accesses the resource) in terms of groups/levels. For example in a typical home building, we assign each floor as a group. Suppose for a three floor building, the clients such as mobile phone/air conditioner can belong to any of the floor within a building. And we allow various permissions for the clients according to the group it belongs to, as specified in Fig 10.

Client	R	W	U	D
G1	*	-	*	-
G2	*	*	-	-
G3	-	-	-	*

Fig 10. Example Permissions on Methods

Supposed we assigned the priorities for different groups as C belongs to {G1, G2, G3} => {P1, P3, P2}. Moreover, if we would like to assign different QoS classes for clients, depending on the applications they use then it is required to control QoS policies in resource control. QoS is defined in terms of various parameters such as {availability, reliability, serviceability, data accuracy, aggregation delay, coverage, fault tolerance, network lifetime} in wireless sensor networks. It is assumed that based on these parameters, QoS is defined in terms of various classes such as {Q1, Q2, Q3}, then it is required that some of the clients can make some pre-level agreements on QoS requirement for their applications either based on the groups it belongs to or based on the priority of the clients request (Suppose, C belongs to {Q1, Q2, Q3}). Method for defining QoS classes is out of scope of the present document. Once defining the groups, its priorities, QoS classes, and permissions, then the conditional statements which define the resource control policies can be defined as follows:

ST1: If the client belongs to G1 then it is allowed with permissions {R, R/W, U}, priority {P1}, QoS {Q1}, and operations {turn it up, read}; else if the client belongs to G2 then it is allowed with permissions {R, W, R/W}, priority {P3}, QoS {Q2}, and operations {turn it up, read}; else if the client belongs to G3 then it is allowed with permissions {D}, priority {P2}, QoS {Q3}, and operations {turn it down}.

ST2: Allow the client with priority {P1}, QoS {Q1}, operations

{turn it up, turn it down, read}, and allow only with permissions {R} in G1; permissions {R, R/W, D} in G2; and permissions {D} in G3.

ST3: Allow the client with priority {P1}, QoS {Q1}, and allow with permissions {R}, operations {read} in G1; allow with permissions {R, R/W, D}, operations {turn it up, turn it down, read} in G2; and allow with permissions {D}, operations {turn it down} in G3.

Above conditional statements are few examples on how to define the conditional statements, the statements can be defined on any manner based on the resource control policies we would like to achieve. The above statements can be better explained in plain semantic notation as shown in Fig 11(a)-13(a), and the corresponding JSON representations for message exchange is explained in Fig 11(b)-13(b). These statements can be even implemented using data modeling language such as YANG or ASN 1.1 which is out of scope of the present document.

<pre> C { G1 { Allow {R,U} Priority {P1} QoS {Q1} Operations {turn it up, read} } G2 { Allow {R,W} Priority {P3} QoS {Q2} Operations {turn it up, read} } G3 { Allow {D} Priority {P2} QoS {Q2} Operations {turn it down} } } </pre>	<pre> "[" "C":{"G1":{"Allow":"R,U", "Priority":"P1","QoS":"Q1", "Operations":"turnup,read"}, "G2":{"Allow":"R,W", "Priority":"P3","QoS":"Q2", "Operations":"turn it up,read"},"G3":{"Allow":"D", "Priority":"P2","QoS":"Q3", "Operations":"turn it down" }}]" </pre>
(a)	(b)

Fig 11. ST1: (a) Semantic Notation (b) JSON Representation

<pre> C { Priority {P1} QoS {Q1} Operations {turn it up,turn it down, read} G1 { Allow {R} }; G2 { Allow {R,W,D} }; G3 { Allow {D} }; } </pre>	<pre> "[" "Priority":"P1","QoS":"Q1", "Operations":"turn it up, turn it down, read", "C":{"G1":{"Allow":"R"}, "G2":{"Allow":"R,W,D"}, "G3":{"Allow":"D"}}]" </pre>
---	---

(a)

(b)

Fig 12. ST2: (a) Semantic Notation (b) JSON Representation

<pre> C { Priority {P1} QoS {Q1} G1 { Allow {R} Operations {read} }; G2 { Allow {R,W,D} Operations {turn it up, turn down, read} }; G3 { Allow {D} Operations {turn it down} }; } </pre>	<pre> "[" "Priority":"P1","QoS": "Q1","C":{"G1": {"Allow": "R","Operations":"read"}, "G2":{"Allow":"R,W,D", "Operations":"turn it up, turn it down, read"}, "G3":{"Allow":"D", "Operations":"turn it down"}}}] " </pre>
--	---

(a)

(b)

Fig 13. ST3: (a) Semantic Notation (b) JSON Representation

6.4 Search for services by device

Any client device (as explained for scenario 2) MUST interact with the provisioning server and looks for deployed services by devices. Moreover, the provisioning server can verify the complete authorization, admission, and resource control of any device's services. Whereas, if any other constrained devices (ex. air conditioner) searches for services hosted by other constrained device (as explained for scenario 1) MUST interact with the RD server as shown in Fig 10. Here, initially the device queries for all services that are hosted by other devices, then it searches within the domain for specific service, its SRV info, and path to the hosted service. Before sending a request, it MUST establish a secure channel between these two nodes [draft-schmitt-ace-twowayauth-for-iot].

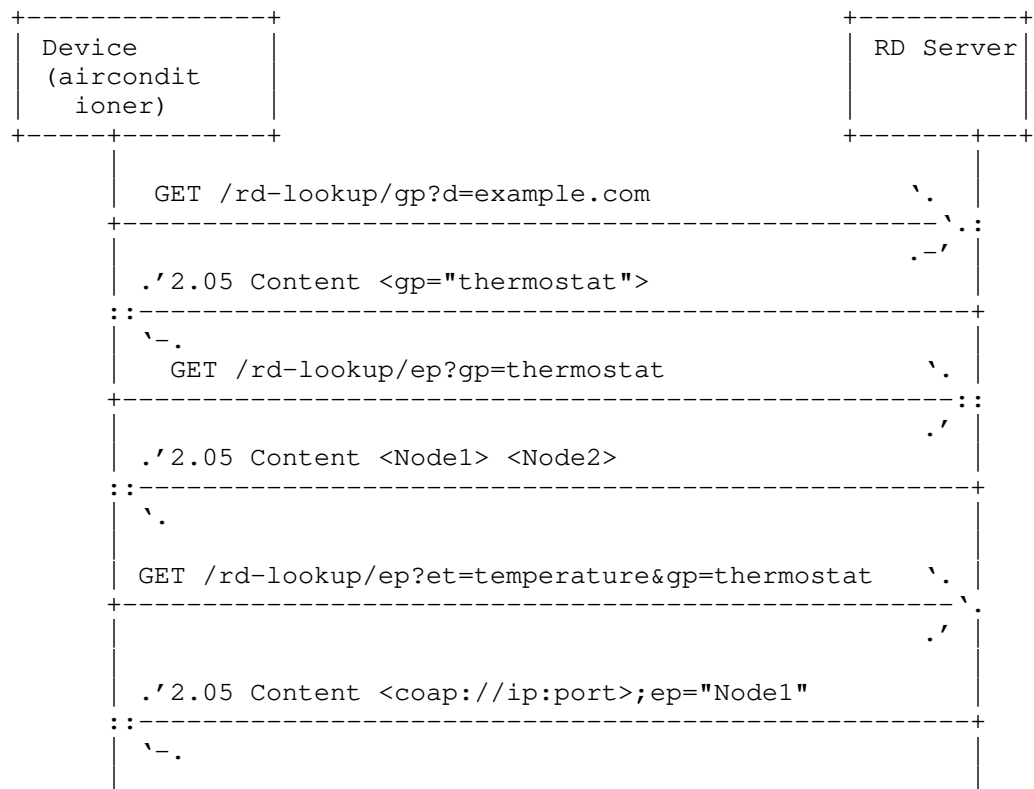


Fig. 10 Search for services by device

6.5 Service request and response

In scenario 1 (as shown in Fig 1), service request and response MUST use coap based communication to access the service as shown in Fig 11. Before sending a request, it MUST establish a secure channel between these two nodes [draft-schmitt-ace-twowayauth-for-iot]. Suppose, the constrained client device (for ex. airconditioner) want to access the service hosted by another constrained device (for ex. thermostat), then the client device MUST send a coap based GET request to thermostat. Then, this device (thermostat) SHOULD send a POST request to provision this service request with the provisioning server by sending clients <IP:port>. Based on the clients <IP:port>, the provisioning server MUST find the client (ex. airconditioner) details such as service information, group, domain, and type details.

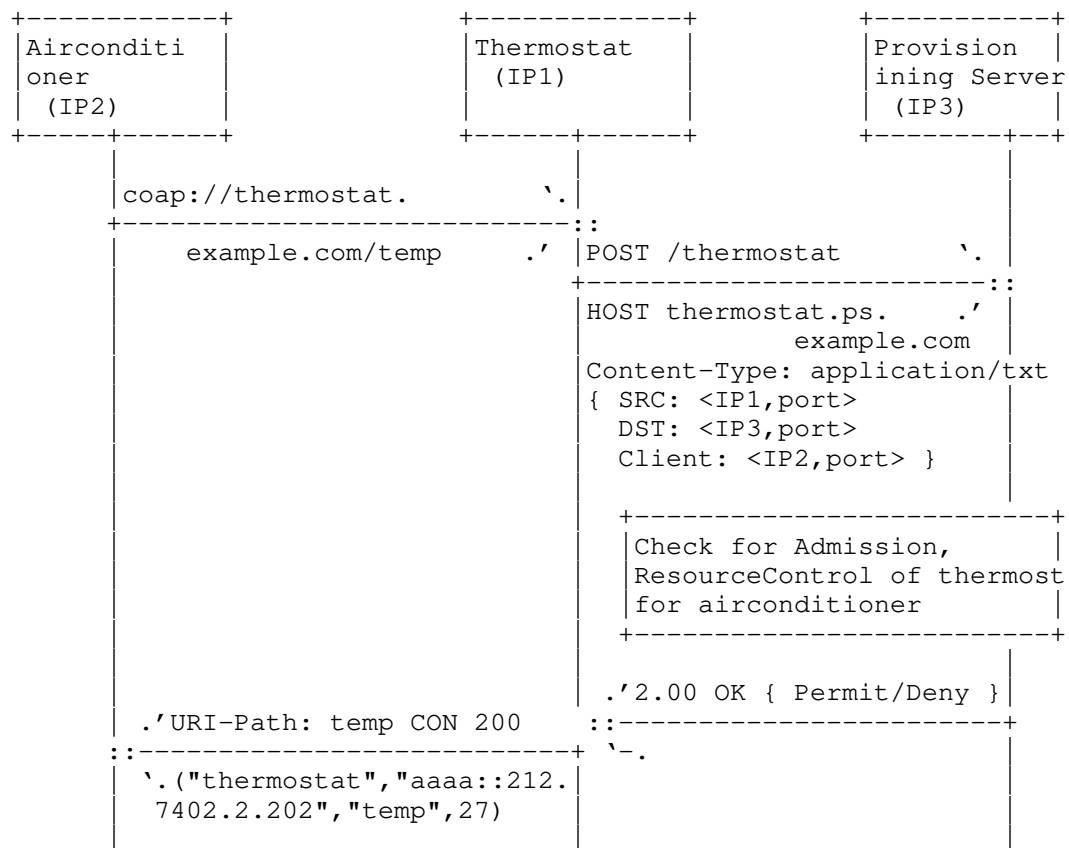


Fig. 11 Request/Response within Constrained Environment

Once the client is identified, the provisioning server MUST check for authorization, admission and resource control policies of

hosted service (ex. thermostat). Once the service request is authorized to access then the URI-Path for hosted service along with the value is sent as a coap response to client device (air conditioner). Here, the request is conditional i.e. based on the resource control policies of a resource (such as thermostat) for a client (airconditioner), the permissions are given to access the resource.

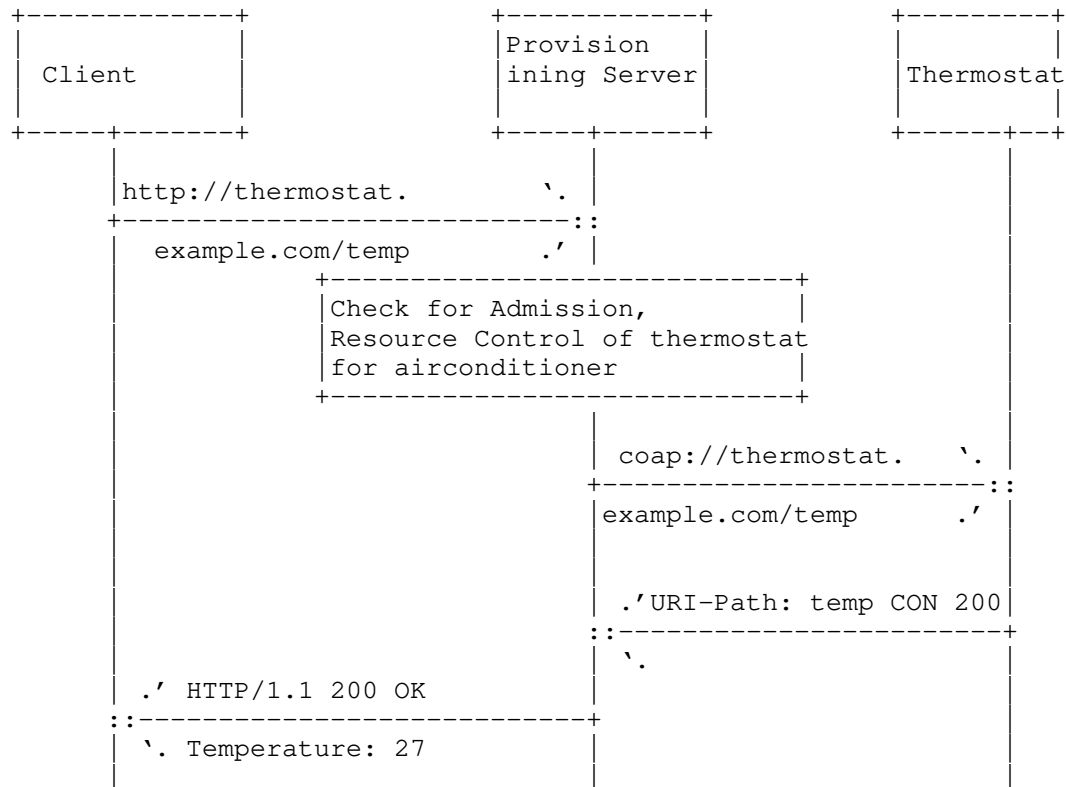


Fig. 12 Request/Response from outside Constrained Environment

Service request and response in scenario 2 (as shown in Fig 2), uses simple http based communication to access the service from the PS. Provisioning Server then sends a coap based GET request to the ultimate device that hosts service. Before sending this request to the actual device for service, PS authorizes the service request. Once, the service request is authorized to access, then the URI-path for hosted service along with the value is sent as HTTP response to client device. PS can implement a reverse proxy case for HTTP-CoAP protocol translation defined in

[draft-ietf-core-http-mapping].

```
-----HTTP begin -----
HTTP POST
Request:
POST /thermostat /HTTP/1.1
HOST thermostat.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
licenseID=string & content=string & paramsXML=string

Response:
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://xyz.com/">
string
</string>

-----HTTP end -----

-----REST via HTTP begin -----
REST via HTTP POST
Request:
POST /thermostat /HTTP/1.1
HOST thermostat.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length

licenseID=string & content=string & paramsXML=string

Response:
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

string

-----REST via HTTP end -----

-----SOAP begin -----

SOAP 1.2
Request:
POST /Thermostat /HTTP/1.1
HOST: www.example.org
```

Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

```
<?xml version="1.0"?>
<soap:envelop>
Xmlns:soap=http://www.w3.org/2001/12/soap-envelop
Soap:encodingStyle=http://www.w3.org/2001/12/soapencoding>
<soap:body xmlns: m="http://www.myhome.org/thermostat">
<m:GetTemperature>
<m:thermostat>1</m:thermostat>
</m:GetTemperature>
</soap:body>
</soap:envelop>
```

Response:
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

```
<?xml version="1.0"?>
<soap:envelop>
Xmlns:soap=http://www.w3.org/2001/12/soap-envelop
Soap:encodingStyle=http://www.w3.org/2001/12/soapencoding>
<soap:body xmlns: m="http://www.example.org/thermostat">
<m:GetTemperatureResponse>
<m:temperature>27.8</m:temperature>
</m:GetTemperatureResponse>
</soap:body>
</soap:envelop>
```

-----SOAP end -----

7 Security Considerations

Security level for message authentication is out of scope of the present document. However, the following security consideration needs to be considered for the present proposed method. Services that run over UDP are unprotected and vulnerable to unknowingly become part of a DDoS attack as UDP does not require return routability check. Therefore, an attacker can easily spoof the source IP of the target entity and send requests to such a service which would then respond to the target entity. The TLS/DTLS based security solution can be considered for secure message communication.

8 IANA Considerations

TBD

9 References

9.1 Normative References

9.2 Informative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [draft-ietf-core-resource-directory] Shelby, Z., and Bormann, C., "CoRE Resource Directory", draft-ietf-core-resource-directory-02 (work in progress), November 2014.
- [draft-gerdes-ace-actors] Gerdes, S., "Actors in the ACE Architecture", draft-gerdes-ace-actors-03 (work in progress), March 2015.
- [draft-gerdes-ace-dcaf-authorize] Gerdes, S., Bergmann, O., Bormann, C., "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-02, March 2015.
- [draft-bormann-core-ace-aif] Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-oo, January 2014.
- [draft-schmitt-ace-twowayauth-for-iot] Schmitt, C., Stiller, B., "Two-way Authentication for IoT", draft-schmitt-ace-twowayauth-for-iot-01, December 2014.
- [DTLS] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 6347, January 2012.

[TLS] Dierks, T. and C. Allen, "The TLS Protocol Version 1.2", RFC 5246, August 2008.

[draft-ietf-core-http-mapping] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and Dijk, E., "Guidelines for HTTP-CoAP Mapping Implementations", draft-ietf-core-http-mapping-05, (work in progress), Oct 2015.

10 Acknowledgements

Special thanks to Amit Kumar S, Zhengfei, Fubaicheng, Yangjun, Vijayachandran Mariappan, Shashidhar C Shekar, Jayaraghavendran K, Ajay Sankar, Puneet Balmukund Sharma, and Rabi Narayan Sahoo for extensive comments and contributions that improved the text.

Thanks to Hedanping (Ana), Behcet Sarikaya, and Carsten Bormann for helpful comments and discussions that have shaped the document.

Authors' Addresses

Vasu K
Huawei Technologies
Bangalore
India

EMail: vasu.kantubukta@huawei.com

Rahul A Jadhav
Huawei Technologies
Bangalore
India

EMail: rahul.jadhav@huawei.com

yangneng
Huawei Technologies
Shenzhen
China

EMail: yangneng@huawei.com

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: September 12, 2016

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
A. Somaraju
Tridonic GmbH & Co KG
R. Turner
Landis+Gyr
A. Minaburo
Acklio
March 11, 2016

Constrained Objects Language
draft-veillette-core-cool-01

Abstract

This document describes a management function set adapted to constrained devices and constrained networks (e.g., low-power, lossy). CoOL objects (datastores, RPCs, actions and notifications) are defined using the YANG modelling language [I-D.ietf-netmod-rfc6020bis]. Interactions with these objects are performed using the CoAP web transfer protocol [RFC7252]. Payloads are encoded using the CBOR data format [RFC7049]. The mapping between YANG data models and the CBOR data format is defined in [I-D.veillette-core-yang-cbor-mapping].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology and Notation	3
3. Architecture	5
4. Resources	5
5. Operations	7
5.1. GET - Retrieving all data nodes of a datastore	7
5.2. FETCH - Retrieving specific data nodes	8
5.2.1. Example #1 - Simple data node	9
5.2.2. Example #2 - Data node instance within a YANG list	11
5.2.3. Example #3 - YANG list	11
5.2.4. Example #4 - YANG list instance	12
5.2.5. Example #5 - YANG list instance filtering	12
5.2.6. Example #6 - All instances of a data node within a YANG list	13
5.3. PUT - Updating all data nodes of a datastore	14
5.4. PATCH - Updating specific data nodes	15
5.5. POST - Protocol operation	17
5.5.1. Example #1 - RPC	17
5.5.2. Example #2 - Action	18
5.6. Event stream	19
6. CoAP compatibility	22
6.1. Working with Uri-Host, Uri-Port, Uri-Path, and Uri-Query	22
6.2. Working with Location-Path and Location-Query	22
6.3. Working with Accept	22
6.4. Working with Max-Age	22
6.5. Working with Proxy-Uri and Proxy-Scheme	22
6.6. Working with If-Match, If-None-Match and ETag	23
6.7. Working with Size1, Size2, Block1 and Block2	23
6.8. Working with resource discovery	24
7. Error Handling	25
8. Security Considerations	26

9. IANA Considerations	26
9.1. "FETCH" CoAP Method Code	26
9.2. "PATCH" CoAP Method Code	26
10. Acknowledgments	26
11. References	27
11.1. Normative References	27
11.2. Informative References	28
Appendix A. File "ietf-cool.yang"	29
Appendix B. File "ietf-cool@2016-01-01.sid"	35
Authors' Addresses	38

1. Introduction

This document defines a CoAP function set for accessing YANG defined resources. YANG data models are encoded in CBOR based on the mapping rules defined in [I-D.veillette-core-yang-cbor-mapping]. YANG items are identified using a compact identifier called Structured Identifiers (SIDs) as defined in [I-D.somaraju-core-sid].

The resulting protocol based on CoAP, CBOR encoded data and structured identifiers (SID) has a low implementation footprint and low network bandwidth requirements and is suitable for both constrained devices and constrained networks as defined by [RFC7228]. This protocol is applicable to the different management topology options described by [I-D.ersue-constrained-mgmt]; centralized, distributed and hierarchical.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in [I-D.ietf-netmod-rfc6020bis]:

- o action
- o data node
- o data tree
- o module
- o notification
- o RPC
- o schema node

- o schema tree
- o submodule

This specification also makes use of the following terminology:

- o candidate configuration datastore: A configuration datastore that can be manipulated without impacting the device's current configuration and that can be committed to the running configuration datastore. Not all devices support a candidate configuration datastore.
- o CoOL client: The originating endpoint of a request, and the destination endpoint of a response.
- o CoOL server: The destination endpoint of a request, and the originating endpoint of a response.
- o delta: Within a list, a delta represents the difference between the current SID and the SID of the previous entry within this list. Within a collection, a delta represents the difference between the SID assigned to the current schema node and the SID assigned to the parent. When no previous entry or parent exist, the delta is set to the absolute SID value.
- o child: A schema node defined within a collection such as a container, a list, a case, a notification, a RPC input, a RPC output, an action input, an action output.
- o datastore: Resource used to store and access information.
- o endpoint: An entity participating in the CoOL protocol. Multiple CoOL endpoints may be accessible using a single CoAP endpoint. In this case, each CoOL endpoint is accessed using a distinct URI.
- o event stream: Resource used to access notifications generated by a CoOL server. Events are defined using the YANG notification statement.
- o function set: A group of well-known resources that provides a particular service.
- o object: Within CoOL, an object is a data node, an RPC or an action within a datastore resource or a notification within an event stream resource.
- o parent: The collection in which a schema node is defined.

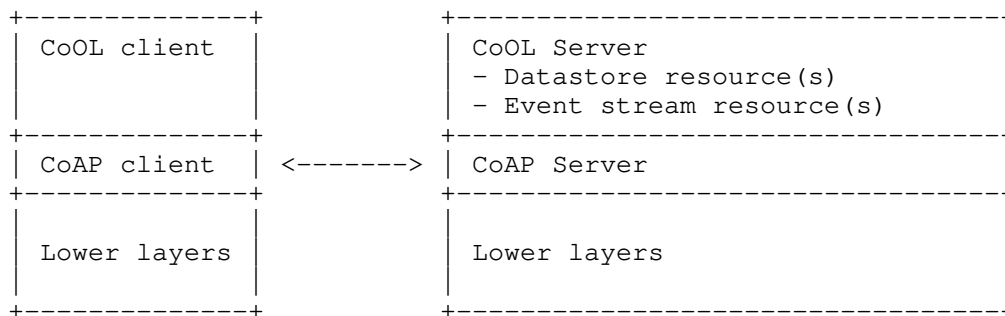
- o resource: Content identified by a URI.
- o running configuration datastore: A configuration datastore holding the complete configuration currently active on the device. The running configuration datastore always exists.
- o Structured Identifier (SID). Unsigned integer used to identify different YANG items.

3. Architecture

The CoOL protocol is based on the client-server model. The CoOL server is the provider of the datastore resource(s) and the event stream resource(s). The CoOL client is the requester of these resources.

CoOL objects are defined using the YANG modeling language [RFC6020]. Interactions with these objects are performed using the Constrained Application Protocol (CoAP) [RFC7252]. Payloads are encoded using the Concise Binary Object Representation (CBOR) [RFC7049].

This specification is applicable to any transport and security protocols supported by CoAP. Implementers are free to select the most appropriate transport for the targeted applications.



4. Resources

This section lists the URIs recommended for the different CoOL resources. A CoOL server MAY implement a different set of URIs. See the Resource discovery section (Section 7.15) for more details on how a CoOL client can discover the list of URIs supported by a CoOL server using the `"/.well-known/core"` resource.

- o `/c` - The default datastore resource
- o `/c/c` - The candidate configuration datastore resource

- o /c/r - The running configuration datastore resource
- o /c/b - The backup configuration datastore (use to implement rollbacks)
- o /c/e - URI used to access the default event stream for this device.
- o /c/e0, /c/e1, ... - URI used to access alternate event streams.
- o /c/0, /c/1, ... - URI used to access a specific endpoint. Each end point represents a virtual device which can support any of the resources listed above.

For example:

- o /c/1 is the default datastore resource for endpoint 1
- o /c/1/c is the candidate datastore resource for endpoint 1
- o /c/1/r is the running configuration datastore resource for endpoint 1
- o /c/1/b is the backup configuration datastore resource for endpoint 1
- o /c/1/e is the default event stream resource for endpoint 1
- o /c/1/e0 is an alternate event stream resource for endpoint 1

All these resources are optional at the exception of the default datastore resource. The CoAP response code 4.04 (Not Found) MUST be returned when a CoOL client tries to access a resource that is unavailable.

RPCs commit and cancel-commit defined in ietf-cool YANG module are available to perform the following operations on datastores:

- o Immediate or differed commit of a candidate or backup datastore.
- o Confirmed commit
- o Cancel of a different or confirmed commit.

5. Operations

This section defines the different interactions supported between a CoOL client and a CoOL server.

5.1. GET - Retrieving all data nodes of a datastore

The GET method is used by CoOL clients to retrieve the entire contents of a datastore. Implementation of this function is optional and dependent of the capability of the CoOL server to transfer a relatively large response.

To retrieve all instantiated data nodes of a datastore resource, a CoOL client sends a CoAP GET request to the URI of the targeted datastore. If the request is accepted by the CoOL server, a 2.05 (Content) response code is returned. The payload of the GET response MUST carry a CBOR array containing the contents of the targeted datastore. The CBOR array MUST contain a list of pairs of delta and associated value. A delta represents the difference between the current SID and the SID of the previous pair within the CBOR array. Each value is encoded using the rules defined by [I-D.veillette-core-yang-cbor-mapping].

If the request is rejected by the CoOL server, a 5.01 Not implemented or 4.13 Request Entity Too Large response code is returned.

Example:

In this example, the CoOL server returns a datastore containing the following data nodes defined in the YANG module "ietf-system" [RFC7317] and YANG module "ietf-interfaces" [RFC7223]:

- o "/interfaces/interface" (SID 1529)
- o "/interfaces/interface/description" (SID 1530)
- o "/interfaces/interface/enabled" (SID 1531)
- o "/interfaces/interface/name" (SID 1533)
- o "/interfaces/interface/type" (SID 1534)
- o "/system-state/clock" (SID 1708)
- o "/system-state/clock/boot-datetime" (SID 1709)
- o "/system-state/clock/current-datetime" (SID 1710)

- o `"/system/clock/timezone/timezone-utc-offset/timezone-utc-offset"`
(SID 1721)

CoAP Request:

GET /c

CoAP response:

2.05 Content Content-Format(application/cool+cbor)

```
[
  1529,
  {
    4 : "eth0",           # name (SID 1533)
    1 : "Ethernet adaptor", # description (SID 1530)
    5 : 1179,             # type (SID 1534), identity ethernetCsmacd
    2 : true              # enabled (SID 1531)
  },
  179,                    # clock (SID 1708)
  {
    1 : "2015-02-08T14:10:08Z09:00", # boot-datetime (SID 1709)
    2 : "2015-04-04T09:32:51Z09:00"  # current-datetime (SID 1710)
  }
  13, 60                  # timezone-utc-offset (SID 1721)
]
```

5.2. FETCH - Retrieving specific data nodes

The FETCH method is used by the CoOL client to retrieve a subset of the data nodes within a datastore.

To retrieve a list of data node instances, the CoOL client sends a CoAP FETCH request to the URI of the targeted datastore. The payload of the FETCH request contains the list of data node(s) instance to be retrieved. This list is encoded using a CBOR array, each entry containing an "instance-identifier" as defined by [I-D.veillette-core-yang-cbor-mapping]. Within each "instance-identifier", data nodes are identified using SIDs as defined by [I-D.somaraju-core-sid].

SIDs within the list of "instance-identifier" are encoded using delta. A delta represents the different between the current SID and the SID of the previous entry within this list. The delta of the first entry within the list is set to the absolute SID value (current SID minus zero).

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.05 (Content).

When a single data node is requested, the payload of the GET response MUST carry the data node instance requested encoded using the rules defined in [I-D.veillette-core-yang-cbor-mapping].

When a multiple data nodes are requested, the payload of the GET response MUST carry a CBOR array containing the data node instance(s) requested. Each entry within this array MUST be encoding using the rules defined in [I-D.veillette-core-yang-cbor-mapping].

When a collection is returned (YANG container, YANG list or YANG list instance), delta(s) are computed using the requested SID as parent.

The CBOR value undefined (0xf7) must be returned for each data node requested but not currently available.

5.2.1. Example #1 - Simple data node

In this example, a CoOL client retrieves the leaf `"/system-state/clock/current-datetime"` (SID 1704) and the container `"/system/clock"` (SID 1719) containing the leaf `"/system/clock/timezone/timezone-utc-offset/timezone-utc-offset"` (SID 1721). These data nodes are defined in the YANG module `"ietf-system"` [RFC7317].

CoAP request:

```
FETCH /c Content-Format(application/cool+cbor)
[1704, 15]
```

CoAP response:

```
2.05 Content Content-Format(application/cool+cbor)
[
  "2015-10-08T14:10:08Z09:00",      # current-datetime (SID 1704)
  {                                # clock (SID 1719)
    2 : 540                        # timezone-utc-offset (SID 1721)
  }
]
```

CoAP requests and responses MUST be encoded in accordance with [RFC7252] or [I-D.ietf-core-coap-tcp-tls]. An encoding example is shown below:

CoAP request:

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|Ver| T |  TKL |  Code (0x01) |           Message ID           |
+-----+-----+-----+-----+
|  Token (0 to 8 bytes) ...  |
+-----+-----+-----+-----+
| Opt Delta (12) | Opt Length (1) |      na      | Opt Delta (3) |
+-----+-----+-----+-----+
| Opt Length (2) |      '/'      |      'c'      | 1 1 1 1 1 1 1 1 |
+-----+-----+-----+-----+
|      0x82      |      0x19      |      0x06      |      0xa8      |
+-----+-----+-----+-----+
|      0x0f      |
+-----+-----+-----+-----+

```

CoAP response:

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|Ver| T |  TKL |  Code (0x45) |           Message ID           |
+-----+-----+-----+-----+
|  Token (0 to 8 bytes) ...  |
+-----+-----+-----+-----+
| Opt Delta (12) | Opt Length (1) |      na      | 1 1 1 1 1 1 1 1 |
+-----+-----+-----+-----+
|      0xa2      |      0x78      |      0x19      |      0x32      |
+-----+-----+-----+-----+
|      0x30      |      0x31      |      0x35      |      0x2d      |
+-----+-----+-----+-----+
|      0x31      |      0x30      |      0x2d      |      0x30      |
+-----+-----+-----+-----+
|      0x38      |      0x54      |      0x31      |      0x34      |
+-----+-----+-----+-----+
|      0x3a      |      0x31      |      0x30      |      0x2d      |
+-----+-----+-----+-----+
|      0x30      |      0x38      |      0x5a      |      0x30      |
+-----+-----+-----+-----+
|      0x39      |      0x3a      |      0x30      |      0x30      |
+-----+-----+-----+-----+
|      0xa1      |      0x02      |      0x19      |      0x02      |
+-----+-----+-----+-----+
|      0x1c      |
+-----+-----+-----+-----+

```


5.2.2. Example #2 - Data node instance within a YANG list

The data type "instance-identifier" allows the selection of an instance of a specific data node within a list. In this example, a CoOL client retrieves the "/interfaces/interface/type" (SID 1529) leaf from the "/interfaces/interface" list. The "/interfaces/interface/name" associated to this interface is equal to "eth0". This example is based on the YANG module "ietf-interfaces" [RFC7223].

CoAP request:

```
FETCH /c Content-Format(application/cool+cbor)
[[1529, "eth0"]]
```

CoAP response:

```
2.05 Content Content-Format(application/cool+cbor)
"Ethernet adaptor"
```

5.2.3. Example #3 - YANG list

To retrieve all instances of a list, the CoOL client excludes from the "instance-identifier" the key(s) of the targeted list. The list returned is encoded using the rules defined in [I-D.veillette-core-yang-cbor-mapping] section 4.4.

In this example, a CoOL client retrieves the list "/interfaces/interface" (SID 1529). The response returns contain two instances, one for an Ethernet adaptor and one for a WIFI interface.

CoAP request:

```
FETCH /c Content-Format(application/cool+cbor)
[1529]
```

CoAP response:

2.05 Content Content-Format(application/cool+cbor)

```
[
  {
    4 : "eth0",           # name (SID 1533)
    1 : "Ethernet adaptor", # description (SID 1530)
    5 : 1179,             # type (SID 1534), identity ethernetCsmacd
    2 : true              # enabled (SID 1531)
  },
  {
    4 : "wlan0",          # name (SID 1533)
    1 : "WIFI ",          # description (SID 1530)
    5 : 1220,             # type (SID 1534), identity ieee80211
    2 : false             # enabled (SID 1531)
  }
]
```

5.2.4. Example #4 - YANG list instance

To retrieve a list instance, the CoOL client **MUST** use an "instance-identifier" with a SID set to the targeted list and the key(s) set to the value(s) associated to the targeted instance.

In this example, the CoOL client requests the instance of the list `"/interfaces/interface"` (SID 1529) associated to the name `"eth1"`. The response returned by the CoOL server contains the targeted list instance formatted as YANG container.

CoAP request:

```
FETCH /c Content-Format(application/cool+cbor)
[[1529, "eth1"]]
```

CoAP response:

2.05 Content Content-Format(application/cool+cbor)

```
{
  4 : "eth0"           # name (SID 1533)
  1 : "Ethernet adaptor" # description (SID 1530)
  5 : 1179             # type (SID 1534), identity ethernetCsmacd
  2 : true             # enabled (SID 1531)
}
```

5.2.5. Example #5 - YANG list instance filtering

This "instance-identifier" extension allows the selection of a subset of data nodes within a list. This is accomplished by adding an extra element to the "instance-identifier". This element contains the subset of data nodes to be returned encoded as CBOR array. Each

entry within this CBOR array is set to the delta between the current SID and the SID of targeted container as specified in the first entry of the "instance-identifier".

CoOL servers SHOULD implement this "instance-identifier" extension. When this extension is not supported, the CoOL server MUST ignore the third element of the "instance-identifier" and return the list instance as specified by the first two elements of the "instance-identifier".

In this example, a CoOL client retrieves from within the "/interfaces/interface" list (SID 1528) the leafs "/interfaces/interface/type" (SID 1533) and "/interfaces/interface/enabled" (SID 1530). The CoOL client also includes in this request the selection of the leaf "/system/hostname" defined in "ietf-system" [RFC7317].

For example:

CoAP request:

```
FETCH /c Content-Format(application/cool+cbor)
[ [1528, ["eth0"], [5, 2]], 211]
```

CoAP response:

```
2.05 Content Content-Format(application/cool+cbor)
[
  {
    5 : 1179,          # type (SID 1533), identity ethernetCsmacd
    2 : true           # enabled (SID 1530)
  },
  "datatracker.ietf.org", # hostname (SID 1739)
]
```

5.2.6. Example #6 - All instances of a data node within a YANG list

This "instance-identifier" extension allows the efficient transfer of all instances of a data node within a YANG list. To retrieve all instances, the CoOL client excludes from the "instance-identifier" the key(s) of the list containing the targeted data node.

The response MUST be encoded as a CBOR ARRAY containing the available instances of the requested data node. This special encoding minimizes significantly this commonly used type of request.

In this example, a CoOL client retrieves all instances of data node "/interfaces-state/interface/name" (SID 1532).

Example:

CoAP request:

```
FETCH /c Content-Format(application/cool+cbor)
[1532]
```

CoAP response:

```
2.05 Content Content-Format(application/cool+cbor)
["eth0", "eth1", "wlan0"]
```

5.3. PUT - Updating all data nodes of a datastore

The CoAP PUT method is used by CoOL clients to update the content of a datastore.

The URI of the PUT request MUST be set to the URI of the targeted datastore.

The payload of the PUT request MUST carry a CBOR array containing the new content of the datastore. The CBOR array MUST contain a list of pairs of delta and associated value. A delta represents the difference between the current SID and the SID of the previous pair within the CBOR array. Each value is encoded using the rules defined by [I-D.veillette-core-yang-cbor-mapping].

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.04 (Changed).

A PUT request MUST be processed as an atomic transaction, if any of the data node transferred is rejected for any reason, the entire PUT request MUST be rejected and the CoOL server MUST return an appropriate error response as defined in section 6.

Example:

In this example, a CoOL client sets the default runtime datastore with these data nodes:

- o `"/system/clock/timezone/timezone-utc-offset/timezone-utc-offset"` (SID 1721)
- o `"/system/ntp/enabled"` (SID 1742)
- o `"/system/ntp/server"` (SID 1743)
- o `"/system/ntp/server/name"` (SID 1746)

- o `"/system/ntp/server/prefer"` (SID 1747)
- o `"/system/ntp/server/transport/udp/udp"` (SID 1748)
- o `"/system/ntp/server/transport/udp/udp/address"` (SID 1749)
- o `"/system/ntp/server/transport/udp/udp/port"` (SID 1750)

CoAP request:

PUT `/c/r Content-Format(application/cool+cbor)`

```
[
  1727, 540,                # timezone-utc-offset (SID 1721)
  15, true,                 # enabled (SID 1742)
  1, [                      # server (SID 1743)
    {
      3 : "tic.nrc.ca",      # name (SID 1746)
      4 : true,              # prefer (SID 1747)
      5 : {                  # udp (SID 1748)
        6 : "132.246.11.231", # address (SID 1749)
        7 : 123              # port (SID 1750)
      }
    },
    {
      3 : "tac.nrc.ca",      # name (SID 1746)
      4 : false,             # prefer (SID 1747)
      5 : {                  # udp (SID 1748)
        6 : "132.246.11.232" # address (SID 1749)
      }
    }
  ]
]
```

CoAP response:

2.04 Changed

5.4. PATCH – Updating specific data nodes

The PATCH method is used by CoOL clients to modify a subset of a datastore.

To modify a datastore, the CoOL client sends a CoAP PATH request to the URI of the targeted datastore. The payload of the FETCH request contains the list of data node instance(s) to be updated, inserted or deleted. This list is encoded using a CBOR array and contains a sequence of pairs of "instance-identifier" and associated values.

Within each "instance-identifier", data nodes are identified using SIDs as defined by [I-D.somaraju-core-sid]. SIDs within the list are encoded as delta.

On reception, the list is processed by the CoOL server as follows:

- o If the targeted data instance already exists, this instance is replaced by the associated value (not merged). To update only some children of a collection, each child data node MUST be provided individually.
- o If the targeted data instance doesn't exist, this instance is created.
- o If the targeted data instance already exists but is associated with the value "null", this instance is deleted.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.05 (Content).

A PATCH request MUST be processed as an atomic transaction, if any of the data nodes transferred is rejected for any reasons, the entire PATCH request MUST be rejected and the CoOL server MUST return an appropriate error response as defined in section 6.

Example:

In this example, a CoOL client performs the following operations:

- o Set "/system/ntp/enabled" to true.
- o Remove the server "tac.nrc.ca" from the "/system/ntp/server" list.
- o Add the server "NTP Pool server 2" to the list "/system/ntp/server".
- o Set "prefer" to false for the server "tic.nrc.ca".

CoAP request:

```

PATCH /c/r Content-Format(application/cool+cbor)
[
  1742 , true,                                # enabled (1742)
  [1, "tac.nrc.ca"], null,                    # server (SID 1743)
  0,                                           # server (SID 1743)
  {
    3 : "NTP Pool server 2",                  # name (SID 1746)
    4 : true,                                # prefer (SID 1747)
    5 : {                                     # udp (SID 1748)
      6 : "2620:10a:800f::11",                # address (SID 1749)
    }
  }
  [4, "tic.nrc.ca"], false                    # prefer (SID 1747)
]

```

CoAP response:

2.04 Changed

5.5. POST - Protocol operation

Protocol operations are defined using the YANG "rpc" or YANG "action" statements.

To execute a protocol operation, the CoOL client sends a CoAP POST request to the URI of the targeted datastore.

The payload of the POST request carries a CBOR array with up to two entries. The first entry carries the instance-identifier identifying the targeted protocol operation. The second entry carries the protocol operation input(s). Input(s) are present only if defined for the invoked protocol operation and used by the CoOL client. Input(s) are encoded using the rules defined for a YANG container, deltas are relative to the SID assigned to the protocol operation.

On successful completion on the protocol operation, the CoOL server returns a CoAP response with the response code set to 2.05 (Content). When output parameters are returned by the CoOL server, these parameter(s) are carried in the CoAP response payload. Output(s) are encoded using the rules defined for a YANG container, deltas are relative to the SID assigned to the protocol operation.

5.5.1. Example #1 - RPC

This example is based on the "activate-software-image" RPC defined in [I-D.ietf-netmod-rfc6020bis], assuming that this RPC is assigned to SID 1932, leaf image-name to SID 1933 and leaf status to SID 1934. These SIDs are defined strictly for the purpose of this example.

```
rpc activate-software-image { input { leaf image-name { type string;
} } output { leaf status { type string; } } }
```

CoAP request:

POST /c Content-Format(application/cool+cbor)

```
[
  1932,
  {
    1 : "acmefw-2.3"                # image-name (SID 1933)
  }
]
```

CoAP response:

2.05 Content

```
{
  2 : "installed"                  # status (SID 1934)
}
```

5.5.2. Example #2 - Action

This example is based on the "reset" action defined in [I-D.ietf-netmod-rfc6020bis] assuming that this action is assigned to SID 1902, leaf reset-at to SID 1903 and leaf reset-finished-at to SID 1904. These SIDs are defined strictly for the purpose of this example.

```
list server { key name; leaf name { type string; } action reset {
input { leaf reset-at { type yang:date-and-time; mandatory true; } }
output { leaf reset-finished-at { type yang:date-and-time; mandatory
true; } } }
```

CoAP request:

POST /c Content-Format(application/cool+cbor)

```
[
  [1902, "myServer"],
  {
    1 : "2016-02-08T14:10:08Z09:00"  # reset-at (SID 1903)
  }
]
```

CoAP response:

2.05 Content

```
{  
  2 : "2016-08T14:10:08Z09:18"      # reset-finished-at (SID 1904)  
}
```

5.6. Event stream

WARNING

This section requires more work to address the following identified issues:

- * Retrieval of past events (e.g. start-time, stop-time)
- * Retrieval of specific events (e.g. filter)
- * Configuration persistence
- * Configuration of by a third entity (configuration tool)
- * Support of multicast
- * Event congestion-avoidance
- * Transfer reliability

The current solution based on the observe CoAP option can be augmented or completely replaced by a future version of this draft.

Notifications are defined using the YANG "notification" statement. Subscriptions to an event stream and notification reporting are performed using an event stream resource. When multiple event stream resources are supported, the list of notifications associated with each stream is either pre-defined or configured in the CoOL server. CoOL clients MAY subscribe to one or more event stream resources.

To subscribe to an event stream resource, a CoOL client MUST send a CoAP GET with the Observe CoAP option set to 0. To unsubscribe, a CoOL client MAY send a CoAP reset or a CoAP GET with the Observe option set to 1. For more information on the observe mechanism, see [RFC7641].

Each notification transferred by a CoOL server to each of the registered CoOL clients is carried in a CoAP response with a response code set to 2.05 (Content). Each CoAP response MUST carry in its payload at least one notification but MAY carry multiple. Each notification is carried in a notification-payload defined in ietf-cool, see Appendix A. The notification-payload supports different meta-data associated to this notification, such as the notification identifier, event timestamp, sequence number, severity level and facility. All of these meta information are optional with the exception of the notification identifier.

The CoAP response payload is encoded using the rules defined for the PUT request. When multiple notifications are reported, the CoAP

response payload carries a CBOR array, with each entry containing a notification.

This example is based on the "link-failure" and "interface-enabled" notifications defined in [I-D.ietf-netmod-rfc6020bis] assuming the following SID assignment:

- o "/link-failure" (SID 1942)
- o "/link-failure/if-name" (SID 1943)
- o "/link-failure/admin-status" (SID 1944)
- o "/interfaces/interface/interface-enabled" (SID 1538)
- o "/interfaces/interface/interface-enabled/by-user" (SID 1539)

These SIDs are defined strictly for the purpose of this example.

```
notification link-failure {
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf admin-status {
    type leafref {
      path "/interface[name = current()/../if-name]/admin-status";
    }
  }
}

container interfaces {
  list interface {
    key "name";

    leaf name {
      type string;
    }

    notification interface-enabled {
      leaf by-user {
        type string;
      }
    }
  }
}
```

In this example, a CoOL client starts by registering to the default event stream resource `"/c/e"`.

CoAP request:

```
GET /c/e observe(0) Token(0x9372)
```

The CoOL server confirms this registration by returning a first CoAP response. The payload of this CoAP response may be empty or may carry the last notification reported by this server.

CoAP response:

```
2.05 Content Observe(52) Token(0xD937)
```

After detecting an event, the CoOL server sends its first notification to the registered CoOL client.

CoAP response:

```
2.05 Content Observe(53) Token(0xD937)
```

```
Content-Format(application/cool+cbor)
```

```
[
  1010 , [1538, "eth0"],           # _id (SID 1010)
  1,{                               # content (SID 1011)
    1 : "bob"                     # by-user (SID 1539)
  }
  5 , "2016-03-08T14:10:08Z09:00", # timestamp (SID 1015)
]
```

To optimize communications or because of other constraints, the CoOL server might transfer multiple notifications in a single CoAP response.

CoAP response:

2.05 Content Observe(52) Token(0xD937)

Content-Format(application/cool+cbor)

```
[
  [
    1010 , [1538, "eth0"],      # _id = interface-enabled (SID 1010)
    1,{                          # content (SID 1011)
      1 : "jack"                # by-user (SID 1539)
    }
    5 , "2016-03-12T15:49:51Z09:00", # timestamp (SID 1015)
  ],
  [
    1010 , 1942,                # _id = link-failure (SID 1010)
    1,{                          # content (SID 1011)
      1 : "eth0",               # if-name (SID 1943)
      1 : 1                     # admin-status = up (SID 1944)
    }
    5 , "2016-03-12T15:50:06Z09:00", # timestamp (SID 1015)
  ]
]
```

6. CoAP compatibility

6.1. Working with Uri-Host, Uri-Port, Uri-Path, and Uri-Query

Uri-Query is not currently used by this protocol. Uri-Host, Uri-Port and Uri-Path MUST be used as specified by [RFC6690] to target the CoOL resources as defined by section 3.

6.2. Working with Location-Path and Location-Query

This version of CoOL doesn't support the creation of resources (datastore or event stream). For this reason, the use of Location-Path and Location-Query is not required.

6.3. Working with Accept

This option is not required since this protocol supports a single content format, "application/cool+cbor".

6.4. Working with Max-Age

This option MUST be supported as specified by [RFC6690].

6.5. Working with Proxy-Uri and Proxy-Scheme

This option MUST be supported as specified by [RFC6690].

6.6. Working with If-Match, If-None-Match and ETag

This option MUST be supported as specified by [RFC6690]. Each ETag is associated to all schema nodes within a datastore.

6.7. Working with Size1, Size2, Block1 and Block2

When the UDP transport is used and a large payload need to be transferred, support of the CoAP block transfer as defined by [I-D.ietf-core-block] is recommended.

6.8. Working with Observe

A CoOL server MAY support state change notifications to some or all its leaf data nodes. When supported the CoOL server MUST implement the Server-Side requirements defined in [RFC7641] section 3 and the CoOL client MUST implement the Client-Side requirements defined in [RFC7641] section 4.

To start observing a leaf data node, a CoOL client MUST send a CoAP FETCH with the Observe CoAP option set to 0.

The payload of the FETCH request carries a CBOR array of instance-identifier. The first entry MUST be set to the "instance-identifier" of the data node instance observed. The following entries are optional and allow the selection of coincidental values, data nodes reported at the same time as the observed data node. Coincidental values are included in each notification reported, but changes to these extra data nodes MUST not trigger notification messages.

A subscription can be terminated by the CoOL client by returning a CoAP Reset message or by sending a GET request with an Observe CoAP option set to deregister (1). More details are available in [RFC7641].

Example:

In this example, a CoOL client subscribes to state changes of the data node `"/system/ntp/enabled"` (SID = 1742) and requests that data node `"/system/hostname"` (SID 1739) is reported as coincidental value.

A first response is immediately returned by the CoOL server to confirm the subscription and to report the current values of the requested data nodes.

Subsequent responses are returned by the CoOL server each time the state of data node `"/system/ntp/enabled"` changes.

CoAP request:

```
FETCH /c Content-Format(application/cool+cbor) Observe(0)
[ [1742, "tic.nrc.ca"], -3 ]
```

CoAP response:

```
2.05 Content Content-Format(application/cool+cbor) Observe(2631)
[
  false,                # enabled (SID 1742)
  "tic"                 # hostname (SID 1739)
]
```

CoAP response:

```
2.05 Content Content-Format(application/cool+cbor) Observe(2632)
[
  true,                 # enabled (SID 1742)
  "tic"                 # hostname (SID 1739)
]
```

6.8. Working with resource discovery

The `"/.well-known/core"` resource is used by CoOL clients to discover resources implemented by CoOL servers. Each CoOL server **MUST** have an entry in the `"/.well-known/core"` resource for each datastore resource and event stream resource supported.

Resource discovery can be performed using a CoAP GET request. If successful, the CoAP response **MUST** have a response code set to 2.05 (Content), a Content-Format set to `"application/link-format"`, and a payload containing a list of web links.

To enable discovery of specific resource types, the CoAP server **MUST** support the query string `"rt"`.

Link format and the `"/.well-known/core"` resource are defined in [RFC6690].

Example:

CoAP request:

```
GET /.well-known/core
```

CoAP response:

```
2.05 Content Content-Format(application/link-format)
</c>;rt="cool.datastore",
</c/r>;rt="cool.datastore",
</c/b>;rt="cool.datastore",
</c/e>;rt="cool.event-stream",
```

In this example, a CoOL client retrieves the list of all resources available on a CoOL server.

Alternatively, the CoOL client may query for a specific resource type. In this example, the CoOL client queries for resource type (rt) "cool.datastore".

CoAP request:

```
GET /.well-known/core?rt=cool.datastore
```

CoAP response:

```
2.05 Content Content-Format(application/link-format)
</c>;rt="cool.datastore",
```

7. Error Handling

All CoAP response codes defined by [RFC7252] MUST be accepted and processed accordingly by CoOL clients. Optionally, client errors (CoAP response codes 4.xx) or server errors (CoAP response codes 5.xx) MAY have a payload providing further information about the cause of the error. This payload contains the "error-payload" container (SID 1006) defined in the "ietf-cool" YANG module, see Appendix A.

Example:

CoAP response:

```
4.00 Bad Request (Content-Format: application/cool+cbor)
[
  1006 , {
    1 : 2,                                # error-code, SID 1007
    2 : "Unknown data node 69687"         # error-text, SID 1008
  }
]
```

8. Security Considerations

This application protocol relies on the lower layers to provide confidentiality, integrity, and availability. A typical approach to archive these requirements is to implement CoAP using the DTLS binding as defined in [RFC7252] section 9. Other approaches are possible to fulfill these requirements, such as the use of a network layer security mechanism as discussed in [I-D.bormann-core-ipsec-for-coap] or a link layer security mechanism for exchanges done within a single sub-network.

In some applications, different access rights to objects (data nodes, protocol operations and notifications) need to be granted to different CoOL clients. Different solutions are possible, such as the implementation of Access Control Lists (ACL) using YANG module(s) or the use of an authorization certificate as defined in [RFC5755]. These access control mechanisms need to be addressed in complementary specifications.

The Security Considerations section of CoAP [RFC7252] is especially relevant to this application protocol and should be reviewed carefully by implementers.

9. IANA Considerations

9.1. "FETCH" CoAP Method Code

This draft makes use of the PATCH CoAP method as defined in [I-D.bormann-core-coap-fetch]. This method needs to be registered in the CoAP Method Codes sub-registry as defined in [RFC7252] section 12.1.1.

9.2. "PATCH" CoAP Method Code

This draft makes use of the PATCH CoAP method as defined in [I-D.vanderstok-core-patch]. This method needs to be registered in the CoAP Method Codes sub-registry as defined in [RFC7252] section 12.1.1.

10. Acknowledgments

This document have been largely inspired by the extensive works done by Andy Bierman and Peter van der Stok on [I-D.vanderstok-core-comi]. [I-D.ietf-netconf-restconf] have also been a critical input to this work. The authors would like to thank the authors and contributors to these two drafts.

The authors would also like to thank Carsten Bormann for his help during the development of this document and his useful comments during the review process.

11. References

11.1. Normative References

- [I-D.bormann-core-coap-fetch]
Bormann, C., "CoAP FETCH Method", draft-bormann-core-coap-fetch-00 (work in progress), October 2015.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.
- [I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-11 (work in progress), February 2016.
- [I-D.vanderstok-core-patch]
Stok, P. and A. Sehgal, "Patch Method for Constrained Application Protocol (CoAP)", draft-vanderstok-core-patch-02 (work in progress), October 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

11.2. Informative References

- [I-D.bormann-core-ipsec-for-coap]
Bormann, C., "Using CoAP with IPsec", draft-bormann-core-ipsec-for-coap-00 (work in progress), December 2012.
- [I-D.ersue-constrained-mgmt]
Ersue, M., Romascanu, D., and J. Schoenwaelder, "Management of Networks with Constrained Devices: Problem Statement, Use Cases and Requirements", draft-ersue-constrained-mgmt-03 (work in progress), February 2013.
- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Technologies, Z., and H. Tschofenig, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-tcp-tls-01 (work in progress), November 2015.
- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-09 (work in progress), December 2015.
- [I-D.vanderstok-core-comi]
Stok, P. and A. Bierman, "CoAP Management Interface", draft-vanderstok-core-comi-09 (work in progress), March 2016.
- [RFC5755] Farrell, S., Housley, R., and S. Turner, "An Internet Attribute Certificate Profile for Authorization", RFC 5755, DOI 10.17487/RFC5755, January 2010, <<http://www.rfc-editor.org/info/rfc5755>>.

- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.

Appendix A. File "ietf-cool.yang"

Module containing the different definitions required by the CoOL protocol.

```
module ietf-cool {
  namespace "urn:ietf:ns:cool";
  prefix cool;

  organization
    "IETF Core Working Group";

  contact
    "Ana Minaburo
    <mailto:ana@ackl.io>

    Abhinav Somaraju
    <mailto:abhinav.somaraju@tridonic.com>

    Alexander Pelov
    <mailto:a@ackl.io>

    Michel Veillette
    <mailto:michel.veillette@trilliantinc.com>

    Randy Turner
    <mailto:Randy.Turner@landisgyr.com>";

  description
    "This module contains the different definitions required
    by the CoOL protocol.";

  revision 2016-01-01 {
    description
      "Initial revision.";
    reference
      "draft-veillette-core-cool";
  }
```

```
// List of useful derived YANG data types for constrained devices

typedef sid {
  type uint32;
  description
    "Structure Identifier value (SID).";
}

typedef utc-time {
  type uint32;
  description
    "Unsigned 32-bit value representing the number of seconds
    since 0 hours, 0 minutes, 0 seconds, on the 1st of January,
    2000 UTC (Universal Coordinated Time).";
}

// Error payload

container error-payload {
  description
    "Optional payload of a client error (CoAP response 4.xx)
    or server error (CoAP response 5.xx).";

  leaf error-code {
    mandatory true;
    type enumeration {
      enum ok {
        value 0;
        description
          "The requested edit have been performed successfully.";
      }

      enum error {
        value 1;
        description "Unspecified error.";
      }

      enum malformed {
        value 2;
        description "Malformed CBOR payload.";
      }

      enum invalid {
        value 3;
        description "The value specified in the request can't be
          apply to the target data node.";
      }
    }
  }
}
```

```
    enum doesNotExist {
        value 4;
        description "The target data node instance specified in
                    the request doesn't exist.";
    }

    enum alreadyExist {
        value 5;
        description "The target data node instance specified in
                    the request already exists.";
    }

    enum readOnly {
        value 6;
        description "Attempt to update a read-only data node.";
    }
}

leaf error-text {
    mandatory false;
    type string;
    description "Textual descriptions of the error.";
}

// Notification payload

identity facility-type {
    description
        "A facility code is used to specify the type of process that
        is logging the message. Notifications from different facilities
        may be handled differently. Other YANG module may add new
        facility type as needed.";
}

identity os {
    base facility-type;
}

identity protocol-stack {
    base facility-type;
}

identity security {
    base facility-type;
}
```

```
identity hardware-monitoring {
  base facility-type;
}

identity application {
  base facility-type;
}

container notification-payload {
  leaf _id {
    mandatory true;
    type instance-identifier;
    description
      "Identifier associated to the notification reported.";
  }

  leaf timestamp {
    mandatory false;
    type utc-time;
    description
      "Event timestamp. Support of this field is optional
      since its not expected that all implementations have
      implement a real time clock and if so, this clock is
      available at all time.";
  }

  leaf sequence-number {
    mandatory false;
    type uint32;
    description
      "Sequence number associated to each event created by CoOL
      server within a specific event stream.";
  }

  leaf severity-level {
    reference "RFC 5424";
    mandatory false;
    type enumeration {
      enum emergency {
        value 0;
        description
          "System is unusable.";
      }
      enum alert {
        value 1;
        description
          "Should be corrected immediately.";
      }
    }
  }
}
```

```
enum critical {
  value 2;
  description
    "Critical conditions.";
}
enum error {
  value 3;
  description
    "Error conditions.";
}
enum warning {
  value 4;
  description
    "May indicate that an error will occur if action is
    not taken.";
}
enum notice {
  value 5;
  description
    "Events that are unusual, but not error conditions.";
}
enum informational {
  value 6;
  description
    "Normal operational messages that require no action.";
}
enum debug {
  value 7;
  description
    "Information useful to developers for debugging the
    application.";
}
}
description
  "Severity associated with this event.";
}

leaf facility {
  mandatory false;
  type identityref {
    base facility-type;
  }
  description
    "Type of process that is logging the message.";
  reference "RFC 5424";
}

leaf content {
```

```
        mandatory false;
        type anydata;
        description
            "Notification container as defined by the notification YANG
            statement.";
    }
}

rpc commit {
    description
        "Used to commit the changes present in a candidate datastore on
        the runtime datastore specify by the URI used to execute this
        operation.";
    input {
        leaf datastore {
            description
                "Path of the datastore resource used as the source of the
                commit operation. When not present, the default candidate
                datastore resource is used.";
            type string;
            mandatory false;
        }

        leaf commit-date-time {
            description
                "When specified, the commit operation is postponed at the
                specified date and time. When not present, the commit is
                performed on reception of this RPC. Supports of this feature
                is optional.";
            type utc-time;
            mandatory false;
        }

        leaf confirm-timeout {
            description
                "When present, a confirming commit MUST be received within
                this period after the start of the commit process.
                A confirming commit is a commit RPC without the
                confirm-timeout field presents. Supports of this feature
                is optional.";
            type string;
            mandatory false;
        }
    }
}

rpc cancel-commit {
    description
```



```
    "Cancels an ongoing scheduled or confirmed commit.";
  }
}
```

Appendix B. File "ietf-cool@2016-01-01.sid"

Following is the ".sid" file generated for the "ietf-cool" YANG module. See [I-D.somaraju-core-sid] for more details on SID and ".sid" file.

```
{
  "assignment-ranges": [
    {
      "entry-point": 1000,
      "size": 100
    }
  ],
  "module-name": "ietf-cool",
  "module-revision": "2016-01-01",
  "items": [
    {
      "type": "identity",
      "assigned": "2016-03-08T21:59:45Z",
      "label": "ietf-cool:application",
      "sid": 1000
    },
    {
      "type": "identity",
      "assigned": "2016-03-08T21:59:45Z",
      "label": "ietf-cool:facility-type",
      "sid": 1001
    },
    {
      "type": "identity",
      "assigned": "2016-03-08T21:59:45Z",
      "label": "ietf-cool:hardware-monitoring",
      "sid": 1002
    },
    {
      "type": "identity",
      "assigned": "2016-03-08T21:59:45Z",
      "label": "ietf-cool:os",
      "sid": 1003
    },
    {
      "type": "identity",
      "assigned": "2016-03-08T21:59:45Z",
      "label": "ietf-cool:protocol-stack",

```

```
    "sid": 1004
  },
  {
    "type": "identity",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "ietf-cool:security",
    "sid": 1005
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/error-payload",
    "sid": 1006
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/error-payload/error-code",
    "sid": 1007
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/error-payload/error-text",
    "sid": 1008
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/notification-payload",
    "sid": 1009
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/notification-payload/_id",
    "sid": 1010
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/notification-payload/content",
    "sid": 1011
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/notification-payload/facility",
```

```
    "sid": 1012
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/notification-payload/sequence-number",
    "sid": 1013
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/notification-payload/severity-level",
    "sid": 1014
  },
  {
    "type": "node",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/notification-payload/timestamp",
    "sid": 1015
  },
  {
    "type": "rpc",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/cancel-commit",
    "sid": 1016
  },
  {
    "type": "rpc",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/commit",
    "sid": 1017
  },
  {
    "type": "rpc",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/commit/input/commit-date-time",
    "sid": 1018
  },
  {
    "type": "rpc",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/commit/input/confirm-timeout",
    "sid": 1019
  },
  {
    "type": "rpc",
    "assigned": "2016-03-08T21:59:45Z",
    "label": "/commit/input/datastore",
```

```
        "sid": 1020
    }
]
}
```

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov (editor)
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: <http://www.landisgyr.com/>

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 19, 2017

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
A. Somaraju
Tridonic GmbH & Co KG
R. Turner
Landis+Gyr
A. Minaburo
Acklio
July 18, 2016

Constrained Objects Language
draft-veillette-core-cool-02

Abstract

This document describes a management function set adapted to constrained devices and constrained networks (e.g., low-power, lossy). CoOL objects (datastores, RPCs, actions and notifications) are defined using the YANG modelling language [I-D.ietf-netmod-rfc6020bis]. Interactions with these objects are performed using the CoAP web transfer protocol [RFC7252]. Payloads are encoded using the CBOR data format [RFC7049]. The mapping between YANG data models and the CBOR data format is defined in [I-D.ietf-core-yang-cbor].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 19, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology and Notation	3
3. Architecture	5
4. Resources	5
5. Operations	7
5.1. GET - Retrieving all data nodes of a datastore	7
5.2. FETCH - Retrieving specific data nodes	8
5.2.1. Example #1 - Simple data node	9
5.2.2. Example #2 - Data node instance within a YANG list	11
5.2.3. Example #3 - YANG list	12
5.2.4. Example #4 - YANG list instance	12
5.2.5. Example #5 - YANG list instance filtering	13
5.2.6. Example #6 - All instances of a data node within a YANG list	14
5.3. PUT - Updating all data nodes of a datastore	14
5.4. iPATCH - Updating specific data nodes	16
5.5. POST - Protocol operation	18
5.5.1. Example #1 - RPC	18
5.5.2. Example #2 - Action	19
5.6. Event stream	19
6. Uri-Query	23
6.1. The 'a' Query Parameter	23
7. CoAP compatibility	24
7.1. Working with Uri-Host, Uri-Port, Uri-Path, and Uri-Query	24
7.2. Working with Location-Path and Location-Query	24
7.3. Working with Accept	24
7.4. Working with Max-Age	24
7.5. Working with Proxy-Uri and Proxy-Scheme	24
7.6. Working with If-Match, If-None-Match and ETag	24
7.7. Working with Sizen1, Sizen2, Block1 and Block2	24
7.8. Working with Observe	24

7.9. Working with resource discovery	26
8. Error Handling	27
9. Security Considerations	27
10. IANA Considerations	28
10.1. CoAP Content-Formats	28
10.2. CBOR simple value	29
11. Acknowledgments	29
12. References	29
12.1. Normative References	29
12.2. Informative References	30
Appendix A. File "ietf-cool.yang"	32
Appendix B. File "ietf-cool@2016-01-01.sid"	38
Authors' Addresses	40

1. Introduction

This document defines a CoAP function set for accessing YANG defined resources. YANG data models are encoded in CBOR based on the mapping rules defined in [I-D.ietf-core-yang-cbor]. YANG items are identified using a compact identifier called Structured Identifiers (SIDs) as defined in [I-D.somaraju-core-sid].

The resulting protocol based on CoAP, CBOR encoded data and structured identifiers (SID) has a low implementation footprint and low network bandwidth requirements and is suitable for both constrained devices and constrained networks as defined by [RFC7228]. This protocol is applicable to the different management topology options described by [I-D.ersue-constrained-mgmt]; centralized, distributed and hierarchical.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in [I-D.ietf-netmod-rfc6020bis]:

- o action
- o data node
- o data tree
- o module
- o notification

- o RPC
- o schema node
- o schema tree
- o submodule

This specification also makes use of the following terminology:

- o candidate configuration datastore: A configuration datastore that can be manipulated without impacting the device's current configuration and that can be committed to the running configuration datastore. Not all devices support a candidate configuration datastore.
- o CoOL client: The originating endpoint of a request, and the destination endpoint of a response.
- o CoOL server: The destination endpoint of a request, and the originating endpoint of a response.
- o delta: Within a list, a delta represents the difference between the current SID and the SID of the previous entry within this list. Within a collection, a delta represents the difference between the SID assigned to the current schema node and the SID assigned to the parent. When no previous entry or parent exist, the delta is set to the absolute SID value.
- o child: A schema node defined within a collection such as a container, a list, a case, a notification, a RPC input, a RPC output, an action input, an action output.
- o datastore: Resource used to store and access information.
- o endpoint: An entity participating in the CoOL protocol. Multiple CoOL endpoints may be accessible using a single CoAP endpoint. In this case, each CoOL endpoint is accessed using a distinct URI.
- o event stream: Resource used to access notifications generated by a CoOL server. Events are defined using the YANG notification statement.
- o function set: A group of well-known resources that provides a particular service.

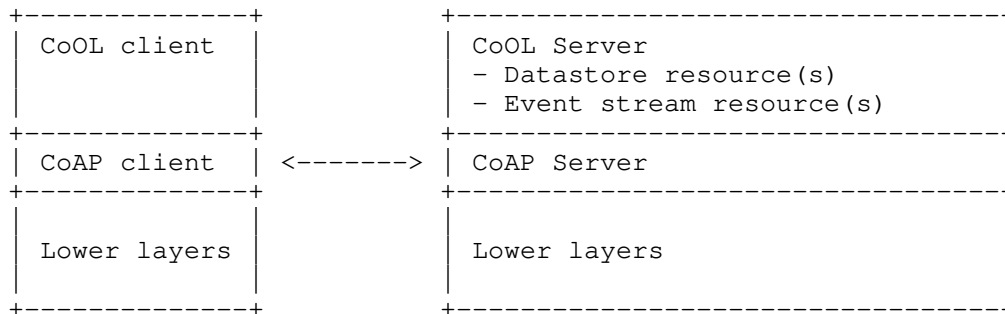
- o **object:** Within CoOL, an object is a data node, an RPC or an action within a datastore resource or a notification within an event stream resource.
- o **parent:** The collection in which a schema node is defined.
- o **resource:** Content identified by a URI.
- o **running configuration datastore:** A configuration datastore holding the complete configuration currently active on the device. The running configuration datastore always exists.
- o **Structured Identifier (SID).** Unsigned integer used to identify different YANG items.

3. Architecture

The CoOL protocol is based on the client-server model. The CoOL server is the provider of the datastore resource(s) and the event stream resource(s). The CoOL client is the requester of these resources.

CoOL objects are defined using the YANG modeling language [RFC6020]. Interactions with these objects are performed using the Constrained Application Protocol (CoAP) [RFC7252]. Payloads are encoded using the Concise Binary Object Representation (CBOR) [RFC7049].

This specification is applicable to any transport and security protocols supported by CoAP. Implementers are free to select the most appropriate transport for the targeted applications.



4. Resources

This section lists the URIs recommended for the different CoOL resources. A CoOL server MAY implement a different set of URIs. See the Resource discovery section (Section 7.15) for more details on how

a CoOL client can discover the list of URIs supported by a CoOL server using the `"/.well-known/core"` resource.

- o `/c` - The default datastore resource
- o `/c/c` - The candidate configuration datastore resource
- o `/c/r` - The running configuration datastore resource
- o `/c/b` - The backup configuration datastore (use to implement rollbacks)
- o `/c/e` - URI used to access the default event stream for this device.
- o `/c/e0`, `/c/e1`, ... - URI used to access alternate event streams.
- o `/c/0`, `/c/1`, ... - URI used to access a specific endpoint. Each end point represents a virtual device which can support any of the resources listed above.

For example:

- o `/c/1` is the default datastore resource for endpoint 1
- o `/c/1/c` is the candidate datastore resource for endpoint 1
- o `/c/1/r` is the running configuration datastore resource for endpoint 1
- o `/c/1/b` is the backup configuration datastore resource for endpoint 1
- o `/c/1/e` is the default event stream resource for endpoint 1
- o `/c/1/e0` is an alternate event stream resource for endpoint 1

All these resources are optional at the exception of the default datastore resource. The CoAP response code 4.04 (Not Found) MUST be returned when a CoOL client tries to access a resource that is unavailable.

RPCs `commit` and `cancel-commit` defined in `ietf-cool` YANG module are available to perform the following operations on datastores:

- o Immediate or differed commit of a candidate or backup datastore.
- o Confirmed commit

- o Cancel of a different or confirmed commit.

5. Operations

This section defines the different interactions supported between a CoOL client and a CoOL server.

5.1. GET - Retrieving all data nodes of a datastore

The GET method is used by CoOL clients to retrieve the entire contents of a datastore. Implementation of this function is optional and dependent of the capability of the CoOL server to transfer a relatively large response.

To retrieve all instantiated data nodes of a datastore resource, a CoOL client sends a CoAP GET request to the URI of the targeted datastore. If the request is accepted by the CoOL server, a 2.05 (Content) response code is returned. The payload of the GET response MUST carry a CBOR array containing the contents of the targeted datastore. The CBOR array MUST contain a list of pairs of delta and associated value. A delta represents the difference between the current SID and the SID of the previous pair within the CBOR array. Each value is encoded using the rules defined by [I-D.ietf-core-yang-cbor].

The normal behaviour of a CoOL server is to exclude from the GET response, any data node currently set to its default value. When this behaviour is not appropriate for the CoOL client, this client can force the retrieval of all data nodes by using the 'a' Uri-Query parameter, see Section 6.1 for more details.

If the request is rejected by the CoOL server, a 5.01 Not implemented or 4.13 Request Entity Too Large response code is returned.

Example:

In this example, the CoOL server returns a datastore containing the following data nodes defined in the YANG module "ietf-system" [RFC7317] and YANG module 'ietf-interfaces' [RFC7223]:

- o "/interfaces/interface" (SID 1533)
- o "/interfaces/interface/description" (SID 1534)
- o "/interfaces/interface/enabled" (SID 1535)
- o "/interfaces/interface/name" (SID 1537)

- o `"/interfaces/interface/1538"` (SID 1534)
- o `"/system-state/clock"` (SID 1717)
- o `"/system-state/clock/boot-datetime"` (SID 1718)
- o `"/system-state/clock/current-datetime"` (SID 1719)
- o `"/system/clock/timezone/timezone-utc-offset/timezone-utc-offset"` (SID 1736)

CoAP Request:

GET /c

CoAP response:

2.05 Content Content-Format(application/cool-value-pairs+cbor)

```
[
  1533,                                # interface (SID 1533)
  {
    +4 : "eth0",                        # name (SID 1537)
    +1 : "Ethernet adaptor",            # description (SID 1534)
    +5 : 1179,                          # type (SID 1538), identity ethernetCsmacd
    +2 : true                            # enabled (SID 1535)
  },
  +184,                                # clock (SID 1717)
  {
    +1 : "2015-02-08T14:10:08Z09:00",   # boot-datetime (SID 1718)
    +2 : "2015-04-04T09:32:51Z09:00"    # current-datetime (SID 1719)
  },
  +19, 60                              # timezone-utc-offset (SID 1736)
]
```

5.2. FETCH - Retrieving specific data nodes

The FETCH method is used by the CoOL client of retrieve a subset of the data node instances within a datastore.

To retrieve a list of data node instances, the CoOL client sends a CoAP FETCH request to the URI of the targeted datastore. The payload of the FETCH request contains the list of data node(s) instance to be retrieved. This list is encoded using a CBOR array, each entry containing an 'instance-identifier' as defined by [I-D.ietf-core-yang-cbor]. Within each 'instance-identifier', data nodes are identified using SIDs as defined by [I-D.somaraju-core-sid].

SIDs within the list of 'instance-identifier' are encoded using delta. A delta represents the different between the current SID and the SID of the previous entry within this list. The delta of the first entry within the list is set to the absolute SID value.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.05 (Content).

When a single data node is requested, the payload of the FETCH response carries the value of the data node instance requested. When multiple data nodes are requested, the payload of the FETCH response carries a CBOR array containing the value of each data node instance(s) requested. The number of entries in this CBOR array MUST match the number of "instance-identifier" requested to allow a proper interpretation of this information. The following values can be returned for each 'instance-identifier' requested:

- o If the data node requested is not implemented or not instantiated, the CBOR simple value 'undefined' is returned.
- o If the URI-Query parameter 'a' is not present in the FETCH request and the value of the data node instance is equal to the default value for this data node, the CBOR simple value 'default' is returned.
- o Otherwise, the data node instance is encoded using the rules defined in [I-D.ietf-core-yang-cbor].

The normal behaviour of a CoOL server is to exclude from containers and list instances of a FETCH response, any data node currently set to its default value. When this behaviour is not appropriate for the CoOL client, this client can force the retrieval of all data nodes by using the 'a' Uri-Query parameter, see Section 6.1 for more details.

5.2.1. Example #1 - Simple data node

In this example, a CoOL client retrieves the leaf "/system-state/clock/current-datetime" (SID 1719) and the container "/system/clock" (SID 1734) containing the leaf "/system/clock/timezone/timezone-utc-offset/timezone-utc-offset" (SID 1736). These data nodes are defined in the YANG module "ietf-system" [RFC7317].

CoAP request:

```
FETCH /c Content-Format(application/cool-instance-id-list+cbor)
[1719, +15]
```

CoAP response:

2.05 Content Content-Format(application/cool-value-list+cbor)

```
[
  "2015-10-08T14:10:08Z09:00",      # current-datetime (SID 1719)
  {                                  # clock (SID 1734)
    +2 : 540                        # timezone-utc-offset (SID 1736)
  }
]
```

CoAP requests and responses MUST be encoded in accordance with [RFC7252] or [I-D.ietf-core-coap-tcp-tls]. An encoding example is shown below:

CoAP request:

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Ver| T |  TKL |  Code (0x05) |           Message ID           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Token (0 to 8 bytes) ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Opt Delta (12) | Opt Length (1) |      na      | Opt Delta (3) |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Opt Length (2) |      '/'      |      'c'      | 1 1 1 1 1 1 1 1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      0x82      |      0x19      |      0x06      |      0xb7      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      0x0f      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

CoAP response:

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Ver T TKL Code (0x45)										Message ID																													
Token (0 to 8 bytes) ...																																							
Opt Delta (12)										Opt Length (1)										na 1 1 1 1 1 1 1 1																			
0x82										0x78										0x19 0x32																			
0x30										0x31										0x35 0x2d																			
0x31										0x30										0x2d 0x30																			
0x38										0x54										0x31 0x34																			
0x3a										0x31										0x30 0x3a																			
0x30										0x38										0x5a 0x30																			
0x39										0x3a										0x30 0x30																			
0xa1										0x02										0x19 0x02																			
0x1c																																							

5.2.2. Example #2 - Data node instance within a YANG list

The data type 'instance-identifier' allows the selection of an instance of a specific data node within a list. In this example, a CoOL client retrieves the "/interfaces/interface/description" (SID 1534) leaf from the "/interfaces/interface" list. The "/interfaces/interface/name" associated to this interface is equal to "eth0". This example is based on the YANG module "ietf-interfaces" [RFC7223].

CoAP request:

```
FETCH /c Content-Format(application/cool-instance-id-list+cbor)
[[1534, "eth0"]]
```

CoAP response:

```
2.05 Content Content-Format(application/cool-value+cbor)
"Ethernet adaptor"
```


5.2.3. Example #3 - YANG list

This "instance-identifier" extension allows the retrieval of all instances of a YANG list. To perform this operation, the CoOL client excludes from the 'instance-identifier' the key(s) of the targeted list. The list returned is encoded using the rules defined in [I-D.ietf-core-yang-cbor] section 4.4.

In this example, a CoOL client retrieves the list `"/interfaces/interface"` (SID 1533). The response returned contain two instances, one for an Ethernet adaptor and one for a WIFI interface.

CoAP request:

```
FETCH /c Content-Format(application/cool-instance-id-list+cbor)
[1533]
```

CoAP response:

2.05 Content Content-Format(application/cool-value+cbor)

```
[
  {
    +4 : "eth0",           # name (SID 1537)
    +1 : "Ethernet adaptor", # description (SID 1534)
    +5 : 1179,             # type (SID 1538), identity ethernetCsmacd
    +2 : true              # enabled (SID 1535)
  },
  {
    +4 : "wlan0",          # name (SID 1537)
    +1 : "WIFI ",          # description (SID 1534)
    +5 : 1220,             # type (SID 1538), identity ieee80211
    +2 : false             # enabled (SID 1535)
  }
]
```

5.2.4. Example #4 - YANG list instance

To retrieve a list instance, the CoOL client MUST use an 'instance-identifier' with a SID set to the targeted list and the key(s) set to the value(s) associated to the targeted instance.

In this example, the CoOL client requests the instance of the list `"/interfaces/interface"` (SID 1533) associated to the name "eth0". The response returned by the CoOL server contains the targeted list instance formatted as YANG container.

CoAP request:

```
FETCH /c Content-Format(application/cool-instance-id-list+cbor)
[[1533, "eth0"]]
```

CoAP response:

```
2.05 Content Content-Format(application/cool-value+cbor)
{
  +4 : "eth0"           # name (SID 1537)
  +1 : "Ethernet adaptor" # description (SID 1534)
  +5 : 1179             # type (SID 1538), identity ethernetCsmacd
  +2 : true              # enabled (SID 1535)
}
```

5.2.5. Example #5 - YANG list instance filtering

This 'instance-identifier' extension allows the selection of a subset of data nodes within a list. This is accomplished by adding an extra element to the 'instance-identifier'. This element contains the subset of data nodes to be returned encoded as CBOR array. Each entry within this CBOR array is set to the delta between the current SID and the SID of targeted container as specified in the first entry of the 'instance-identifier'.

CoOL servers SHOULD implement this 'instance-identifier' extension. When this extension is not supported, the CoOL server MUST ignore the third element of the 'instance-identifier' and return the list instance as specified by the first two elements of the 'instance-identifier'.

In this example, a CoOL client retrieves from within the `"/interfaces/interface"` list (SID 1533) the leafs `"/interfaces/interface/type"` (SID 1538) and `"/interfaces/interface/enabled"` (SID 1535). The CoOL client also includes in this request the selection of the leaf `"/system/hostname"` (SID 1748) defined in `"ietf-system"` [RFC7317].

For example:

CoAP request:

```
FETCH /c Content-Format(application/cool-instance-id-list+cbor)
[ [1533, "eth0", [+5, +2]], +215]
```

CoAP response:

```
2.05 Content Content-Format(application/cool-value-list+cbor)
[
  {
    +5 : 1179,          # type (SID 1538), identity ethernetCsmacd
    +2 : true           # enabled (SID 1535)
  },
  "datatracker.ietf.org", # hostname (SID 1748)
]
```

5.2.6. Example #6 - All instances of a data node within a YANG list

This 'instance-identifier' extension allows the efficient transfer of all instances of a data node within a YANG list. To retrieve all instances, the CoOL client excludes from the 'instance-identifier' the key(s) of the list containing the targeted data node.

The response MUST be encoded as a CBOR ARRAY containing the available instances of the requested data node. This special encoding minimizes significantly this commonly used type of request.

In this example, a CoOL client retrieves all instances of data node `"/interfaces/interface/name"` (SID 1537).

Example:

CoAP request:

```
FETCH /c Content-Format(application/cool-instance-id-list+cbor)
[1537]
```

CoAP response:

```
2.05 Content Content-Format(application/cool-value+cbor)
["eth0", "eth1", "wlan0"]
```

5.3. PUT - Updating all data nodes of a datastore

The CoAP PUT method is used by CoOL clients to update the content of a datastore.

The URI of the PUT request MUST be set to the URI of the targeted datastore.

The payload of the PUT request MUST carry a CBOR array containing the new content of the datastore. The CBOR array MUST contain a list of pairs of delta and associated value. A delta represents the different between the current SID and the SID of the previous pair

within the CBOR array. Each value is encoded using the rules defined by [I-D.ietf-core-yang-cbor].

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.04 (Changed).

A PUT request MUST be processed as an atomic transaction, if any of the data node transferred is rejected for any reason, the entire PUT request MUST be rejected and the CoOL server MUST return an appropriate error response as defined in section 6.

Example:

In this example, a CoOL client sets the default runtime datastore with these data nodes:

- o `"/system/clock/timezone/timezone-utc-offset/timezone-utc-offset"` (SID 1736)
- o `"/system/ntp/enabled"` (SID 1751)
- o `"/system/ntp/server"` (SID 1752)
- o `"/system/ntp/server/name"` (SID 1755)
- o `"/system/ntp/server/prefer"` (SID 1756)
- o `"/system/ntp/server/transport/udp/udp"` (SID 1757)
- o `"/system/ntp/server/transport/udp/udp/address"` (SID 1758)
- o `"/system/ntp/server/transport/udp/udp/port"` (SID 1759)

CoAP request:

```

PUT /c/r Content-Format(application/cool-value-pairs+cbor)
[
  1736, 540,                # timezone-utc-offset (SID 1736)
  +15, true,                # enabled (SID 1751)
  +1, [                     # server (SID 1752)
    {
      +3 : "tic.nrc.ca",    # name (SID 1755)
      +4 : true,            # prefer (SID 1756)
      +5 : {               # udp (SID 1757)
        +1 : "132.246.11.231", # address (SID 1758)
        +2 : 123            # port (SID 1759)
      }
    },
    {
      +3 : "tac.nrc.ca",    # name (SID 1755)
      +4 : false,          # prefer (SID 1756)
      +5 : {               # udp (SID 1757)
        +1 : "132.246.11.232" # address (SID 1758)
      }
    }
  ]
]

```

CoAP response:

2.04 Changed

5.4. iPATCH – Updating specific data nodes

The iPATCH method is used by CoOL clients to modify a subset of a datastore.

To modify a datastore, the CoOL client sends a CoAP PATH request to the URI of the targeted datastore. The payload of the FETCH request contains the list of data node instance(s) to be updated, inserted or deleted. This list is encoded using a CBOR array and contains a sequence of pairs of 'instance-identifier' and associated values.

Within each 'instance-identifier', data nodes are identified using SIDs as defined by [I-D.somaraju-core-sid]. SIDs within the list are encoded as delta.

On reception, the list is processed by the CoOL server as follows:

- o If the targeted data instance already exists, this instance is replaced by the associated value (not merged). To update only some children of a collection, each child data node MUST be provided individually.

- o If the targeted data instance doesn't exist, this instance is created.
- o If the targeted data instance already exists but is associated with the value 'null', this instance is deleted.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.05 (Content).

A iPATCH request MUST be processed as an atomic transaction, if any of the data nodes transferred is rejected for any reasons, the entire iPATCH request MUST be rejected and the CoOL server MUST return an appropriate error response as defined in section 6.

Example:

In this example, a CoOL client performs the following operations:

- o Set `"/system/ntp/enabled"` (SID 1751) to true.
- o Remove the server `"tac.nrc.ca"` from the `"/system/ntp/server"` (SID 1752) list.
- o Add the server `"NTP Pool server 2"` to the list `"/system/ntp/server"` (SID 1752).
- o Set `"/system/ntp/server/prefer"` (SID 1756) to false for the server `"tic.nrc.ca"`.

CoAP request:

```
iPATCH /c/r Content-Format(application/cool-value-pairs+cbor)
[
  1751 , true,                               # enabled (1751)
  [+1, "tac.nrc.ca"], null,                  # server (SID 1752)
  +0,                                         # server (SID 1752)
  {
    +3 : "NTP Pool server 2",                # name (SID 1755)
    +4 : true,                               # prefer (SID 1756)
    +5 : {
      +1 : "2620:10a:800f::11",              # address (SID 1758)
    }
  }
  [+4, "tic.nrc.ca"], false                  # prefer (SID 1756)
]
```

CoAP response:

2.04 Changed

5.5. POST - Protocol operation

Protocol operations are defined using the YANG 'rpc' or YANG 'action' statements.

To execute a protocol operation, the CoOL client sends a CoAP POST request to the URI of the targeted datastore.

The payload of the POST request carries a CBOR array with up to two entries. The first entry carries the instance-identifier identifying the targeted protocol operation. The second entry carries the protocol operation input(s). Input(s) are present only if defined for the invoked protocol operation and used by the CoOL client. Input(s) are encoded using the rules defined for a YANG container, deltas are relative to the SID assigned to the protocol operation.

On successful completion on the protocol operation, the CoOL server returns a CoAP response with the response code set to 2.05 (Content). When output parameters are returned by the CoOL server, these parameter(s) are carried in the CoAP response payload. Output(s) are encoded using the rules defined for a YANG container, deltas are relative to the SID assigned to the protocol operation.

5.5.1. Example #1 - RPC

This example is based on the 'activate-software-image' RPC defined in [I-D.ietf-netmod-rfc6020bis], assuming that this RPC is assigned to SID 1932, leaf image-name to SID 1933 and leaf status to SID 1934. These SIDs are defined strictly for the purpose of this example.

```
rpc activate-software-image { input { leaf image-name { type string;
} } output { leaf status { type string; } } }
```

CoAP request:

```
POST /c Content-Format(application/cool-value-pairs+cbor)
[
  1932,
  {
    +1 : "acmefw-2.3"           # image-name (SID 1933)
  }
]
```

CoAP response:

```
2.05 Content Content-Format(application/cool-value+cbor)
{
  +2 : "installed"                # status (SID 1934)
}
```

5.5.2. Example #2 - Action

This example is based on the 'reset' action defined in [I-D.ietf-netmod-rfc6020bis] assuming that this action is assigned to SID 1902, leaf reset-at to SID 1903 and leaf reset-finished-at to SID 1904. These SIDs are defined strictly for the purpose of this example.

```
list server { key name; leaf name { type string; } action reset {
input { leaf reset-at { type yang:date-and-time; mandatory true; } }
output { leaf reset-finished-at { type yang:date-and-time; mandatory
true; } } } }
```

CoAP request:

```
POST /c Content-Format(application/cool-value-pairs+cbor)
[
  [1902, "myServer"],
  {
    +1 : "2016-02-08T14:10:08Z09:00"    # reset-at (SID 1903)
  }
]
```

CoAP response:

```
2.05 Content Content-Format(application/cool-value+cbor)
{
  +2 : "2016-08T14:10:08Z09:18"        # reset-finished-at (SID 1904)
}
```

5.6. Event stream

WARNING

This section requires more work to address the following identified issues:

- * Retrieval of past events (e.g. start-time, stop-time)
- * Retrieval of specific events (e.g. filter)
- * Configuration persistence
- * Configuration of by a third entity (configuration tool)
- * Support of multicast
- * Event congestion-avoidance
- * Transfer reliability

The current solution based on the observe CoAP option can be augmented or completely replaced by a future version of this draft.

Notifications are defined using the YANG 'notification' statement. Subscriptions to an event stream and notification reporting are performed using an event stream resource. When multiple event stream resources are supported, the list of notifications associated with each stream is either pre-defined or configured in the CoOL server. CoOL clients MAY subscribe to one or more event stream resources.

To subscribe to an event stream resource, a CoOL client MUST send a CoAP GET with the Observe CoAP option set to 0. To unsubscribe, a CoOL client MAY send a CoAP reset or a CoAP GET with the Observe option set to 1. For more information on the observe mechanism, see [RFC7641].

Each notification transferred by a CoOL server to each of the registered CoOL clients is carried in a CoAP response with a response code set to 2.05 (Content). Each CoAP response MUST carry in its payload at least one notification but MAY carry multiple. Each notification is carried in a notification-payload defined in ietf-cool, see Appendix A. The notification-payload supports different meta-data associated to this notification, such as the notification identifier, event timestamp, sequence number, severity level and facility. All of these meta information are optional with the exception of the notification identifier.

The CoAP response payload is encoded using the rules defined for the PUT request. When multiple notifications are reported, the CoAP response payload carries a CBOR array, with each entry containing a notification.

This example is based on the 'link-failure' and 'interface-enabled' notifications defined in [I-D.ietf-netmod-rfc6020bis] assuming the following SID assignment:

- o `"/link-failure"` (SID 1942)

- o `"/link-failure/if-name"` (SID 1943)
- o `"/link-failure/admin-status"` (SID 1944)
- o `"/interfaces/interface/interface-enabled"` (SID 1538)
- o `"/interfaces/interface/interface-enabled/by-user"` (SID 1539)

These SIDs are defined strictly for the purpose of this example.

```
notification link-failure {
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf admin-status {
    type leafref {
      path "/interface[name = current()/../if-name]/admin-status";
    }
  }
}

container interfaces {
  list interface {
    key "name";

    leaf name {
      type string;
    }

    notification interface-enabled {
      leaf by-user {
        type string;
      }
    }
  }
}
```

In this example, a CoOL client starts by registering to the default event stream resource `"/c/e"`.

CoAP request:

```
GET /c/e observe(0) Token(0x9372)
```

The CoOL server confirms this registration by returning a first CoAP response. The payload of this CoAP response may be empty or may carry the last notification reported by this server.

CoAP response:

2.05 Content Observe(52) Token(0xD937)

After detecting an event, the CoOL server sends its first notification to the registered CoOL client.

CoAP response:

2.05 Content Observe(53) Token(0xD937)

Content-Format(application/cool-value-pairs+cbor)

```
[
  1011 , [1538, "eth0"],          # _id (SID 1011)
  +1,{                          # content (SID 1012)
    +1 : "bob"                  # by-user (SID 1539)
  }
  +5 , "2016-03-08T14:10:08Z09:00", # timestamp (SID 1016)
]
```

To optimize communications or because of other constraints, the CoOL server might transfer multiple notifications in a single CoAP response.

CoAP response:

2.05 Content Observe(52) Token(0xD937)

Content-Format(application/cool-value-pairs+cbor)

```
[
  [
    1011 , [1538, "eth0"],          # _id = interface-enabled (SID 1011)
    +1,{                          # content (SID 1012)
      +1 : "jack"                  # by-user (SID 1539)
    }
    +5 , "2016-03-12T15:49:51Z09:00", # timestamp (SID 1016)
  ],
  [
    +1011 , 1942,                  # _id = link-failure (SID 1011)
    +1,{                          # content (SID 1011)
      +1 : "eth0",                # if-name (SID 1943)
      +1 : 1                      # admin-status = up (SID 1944)
    }
    +5 , "2016-03-12T15:50:06Z09:00", # timestamp (SID 1016)
  ]
]
```

6. Uri-Query

6.1. The 'a' Query Parameter

When performing a GET, the normal behaviour of a CoOL server is to exclude from the GET response, data nodes currently set to their default values as defined by the YANG 'default' statement. This behaviour called 'trim' is defined in [RFC6243] section 3.2.

This rule also applies to the FETCH for containers and list instances but not for the root data nodes. To minimize the payload size of the FETCH responses, root data nodes are returned in a CBOR array without associated SID. To keep the symmetry between the FETCH request and the FETCH response, a CBOR content must be returned for each data node requested as follows:

- o a CBOR simple type 'default' is returned for each data node currently set to its default value
- o a CBOR simple type 'undefined value' is returned for each data node not instantiated or not supported
- o otherwise, the value is encoded based on the rules defined in [I-D.ietf-core-yang-cbor]

There are use-cases when a CoOL client will need the default data from the server, for example:

- o The management application often needs a single, definitive, and complete set of configuration values that determine how the networking device works.
- o Documentation about default values can be unreliable or unavailable.
- o Some management applications might not have the capabilities to correctly parse and interpret formal data models.
- o Human users might want to understand the received data without consultation of the documentation.

In all these cases, the CoOL client will need a mechanism to retrieve default data from a CoOL server.

This is accomplished by including the 'a' Uri-Query parameter in the GET or FETCH request. This behaviour called 'report-all' is defined in [RFC6243] section 3.1.

7. CoAP compatibility

7.1. Working with Uri-Host, Uri-Port, Uri-Path, and Uri-Query

Supported Uri-Query parameters are defined in Section 6. Uri-Host, Uri-Port and Uri-Path MUST be used as specified by [RFC6690] to target the CoOL resources as defined by section 3.

7.2. Working with Location-Path and Location-Query

This version of CoOL doesn't support the creation of resources (datastore or event stream). For this reason, the use of Location-Path and Location-Query is not required.

7.3. Working with Accept

This option is not required since this protocol don't support multiple choices of Content-Format.

7.4. Working with Max-Age

This option MUST be supported as specified by [RFC6690].

7.5. Working with Proxy-Uri and Proxy-Scheme

This option MUST be supported as specified by [RFC6690].

7.6. Working with If-Match, If-None-Match and ETag

This option MUST be supported as specified by [RFC6690]. Each ETag is associated to all schema nodes within a datastore.

7.7. Working with Size1, Size2, Block1 and Block2

When the UDP transport is used and a large payload need to be transferred, support of the CoAP block transfer as defined by [I-D.ietf-core-block] is recommended.

7.8. Working with Observe

A CoOL server MAY support state change notifications to some or all its leaf data nodes. When supported the CoOL server MUST implement the Server-Side requirements defined in [RFC7641] section 3 and the CoOL client MUST implement the Client-Side requirements defined in [RFC7641] section 4.

To start observing a leaf data node, a CoOL client MUST send a CoAP FETCH with the Observe CoAP option set to 0.

The payload of the FETCH request carries a CBOR array of instance-identifier. The first entry MUST be set to the 'instance-identifier' of the data node instance observed. The following entries are optional and allow the selection of coincidental values, data nodes reported at the same time as the observed data node. Coincidental values are included in each notification reported, but changes to these extra data nodes MUST not trigger notification messages.

A subscription can be terminated by the CoOL client by returning a CoAP Reset message or by sending a GET request with an Observe CoAP option set to deregister (1). More details are available in [RFC7641].

Example:

In this example, a CoOL client subscribes to state changes of the data node `"/system/ntp/enabled"` (SID = 1751) and requests that data node `"/system/hostname"` (SID 1748) is reported as coincidental value.

A first response is immediately returned by the CoOL server to confirm the subscription and to report the current values of the requested data nodes.

Subsequent responses are returned by the CoOL server each time the state of data node `"/system/ntp/enabled"` changes.

CoAP request:

```
FETCH /c Content-Format(application/cool-instance-id-list+cbor) Observe(0)
[ [1751, "tic.nrc.ca"], -3 ]
```

CoAP response:

```
2.05 Content Content-Format(application/cool-value-pairs+cbor) Observe(2631)
[
  false,                # enabled (SID 1751)
  "tic"                  # hostname (SID 1748)
]
```

CoAP response:

```
2.05 Content Content-Format(application/cool-value-pairs+cbor) Observe(2632)
[
  true,                  # enabled (SID 1751)
  "tic"                  # hostname (SID 1748)
]
```

7.9. Working with resource discovery

The `"/.well-known/core"` resource is used by CoOL clients to discover resources implemented by CoOL servers. Each CoOL server **MUST** have an entry in the `"/.well-known/core"` resource for each datastore resource and event stream resource supported.

Resource discovery can be performed using a CoAP GET request. If successful, the CoAP response **MUST** have a response code set to 2.05 (Content), a Content-Format set to `"application/link-format"`, and a payload containing a list of web links.

To enable discovery of specific resource types, the CoAP server **MUST** support the query string `"rt"`.

Link format and the `"/.well-known/core"` resource are defined in [RFC6690].

Example:

CoAP request:

```
GET /.well-known/core
```

CoAP response:

```
2.05 Content Content-Format(application/link-format)
</c>;rt="cool.datastore",
</c/r>;rt="cool.datastore",
</c/b>;rt="cool.datastore",
</c/e>;rt="cool.event-stream",
```

In this example, a CoOL client retrieves the list of all resources available on a CoOL server.

Alternatively, the CoOL client may query for a specific resource type. In this example, the CoOL client queries for resource type (rt) `"cool.datastore"`.

CoAP request:

```
GET /.well-known/core?rt=cool.datastore
```

CoAP response:

```
2.05 Content Content-Format(application/link-format)
</c>;rt="cool.datastore",
```

8. Error Handling

All CoAP response codes defined by [RFC7252] MUST be accepted and processed accordingly by CoOL clients. Optionally, client errors (CoAP response codes 4.xx) or server errors (CoAP response codes 5.xx) MAY have a payload providing further information about the cause of the error. This payload contains the "error-payload" container (SID 1007) defined in the "ietf-cool" YANG module, see Appendix A.

Example:

CoAP response:

```
4.00 Bad Request (Content-Format: application/cool-value-pairs+cbor)
[
  1007 , {                                     # error-payload (SID 1007)
    +1 : 2,                                   # error-code (SID 1008)
    +2 : "Unknown data node 69687"           # error-text (SID 1009)
  }
]
```

9. Security Considerations

This application protocol relies on the lower layers to provide confidentiality, integrity, and availability. A typical approach to archive these requirements is to implement CoAP using the DTLS binding as defined in [RFC7252] section 9. Other approaches are possible to fulfill these requirements, such as the use of a network layer security mechanism as discussed in [I-D.bormann-core-ipsec-for-coap] or a link layer security mechanism for exchanges done within a single sub-network.

In some applications, different access rights to objects (data nodes, protocol operations and notifications) need to be granted to different CoOL clients. Different solutions are possible, such as the implementation of Access Control Lists (ACL) using YANG module(s) or the use of an authorization certificate as defined in [RFC5755]. These access control mechanisms need to be addressed in complementary specifications.

The Security Considerations section of CoAP [RFC7252] is especially relevant to this application protocol and should be reviewed carefully by implementers.

10. IANA Considerations

10.1. CoAP Content-Formats

This draft introduces the following CoAP Content-Formats. These entries need to be registered in the CoAP Content-Formats Registry as defined in [RFC7252] section 12.3.

First entry:

- o Media type = application/cool-instance-id-list
- o Encoding = CBOR
- o ID = 61
- o Reference = RFC XXXX

Second entry:

- o Media type = application/cool-value
- o Encoding = CBOR
- o ID = 62
- o Reference = RFC XXXX

Third entry:

- o Media type = application/cool-value-list
- o Encoding = CBOR
- o ID = 63
- o Reference = RFC XXXX

Fourth entry:

- o Media type = application/cool-value-pairs+cbor
- o Encoding = CBOR
- o ID = 64
- o Reference = RFC XXXX

RFC Ed.: update XXXX to the RFC number assigned to this draft, update the ID if different than the one allocated, remove this note.

TO DO If this set of Content-Formats is accepted, requirements and description need to be added where appropriate.

10.2. CBOR simple value

This draft introduces the following CBOR simple value. This entry needs to be registered in the Simple Values Registry as defined in [RFC7049] section 7.1.

- o Value = 19
- o Semantics = Default value
- o Reference = RFC XXXX

RFC Ed.: update XXXX to the RFC number assigned to this draft, update the value if different than the one allocated, remove this note.

11. Acknowledgments

This document have been largely inspired by the extensive works done by Andy Bierman and Peter van der Stok on [I-D.vanderstok-core-comi]. [I-D.ietf-netconf-restconf] have also been a critical input to this work. The authors would like to thank the authors and contributors to these two drafts.

The authors would also like to thank Carsten Bormann for his help during the development of this document and his useful comments during the review process.

12. References

12.1. Normative References

- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-21 (work in progress), July 2016.
- [I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-14 (work in progress), June 2016.

[I-D.vanderstok-core-etch]

Stok, P., Bormann, C., and A. Sehgal, "Patch and Fetch Methods for Constrained Application Protocol (CoAP)", draft-vanderstok-core-etch-00 (work in progress), March 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <<http://www.rfc-editor.org/info/rfc6243>>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.

12.2. Informative References

[I-D.bormann-core-ipsec-for-coap]

Bormann, C., "Using CoAP with IPsec", draft-bormann-core-ipsec-for-coap-00 (work in progress), December 2012.

- [I-D.ersue-constrained-mgmt]
Ersue, M., Romascanu, D., and J. Schoenwaelder,
"Management of Networks with Constrained Devices: Problem
Statement, Use Cases and Requirements", draft-ersue-
constrained-mgmt-03 (work in progress), February 2013.
- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K.,
Silverajan, B., and B. Raymor, "CoAP (Constrained
Application Protocol) over TCP, TLS, and WebSockets",
draft-ietf-core-coap-tcp-tls-03 (work in progress), July
2016.
- [I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A.
Minaburo, "CBOR Encoding of Data Modeled with YANG",
draft-ietf-core-yang-cbor-02 (work in progress), July
2016.
- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF
Protocol", draft-ietf-netconf-restconf-15 (work in
progress), July 2016.
- [I-D.somaraju-core-sid]
Somaraju, A., Veillette, M., Pelov, A., Turner, R., and A.
Minaburo, "Structure Identifier (SID)", draft-somaraju-
core-sid-01 (work in progress), July 2016.
- [I-D.vanderstok-core-comi]
Stok, P. and A. Bierman, "CoAP Management Interface",
draft-vanderstok-core-comi-09 (work in progress), March
2016.
- [RFC5755] Farrell, S., Housley, R., and S. Turner, "An Internet
Attribute Certificate Profile for Authorization",
RFC 5755, DOI 10.17487/RFC5755, January 2010,
<<http://www.rfc-editor.org/info/rfc5755>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface
Management", RFC 7223, DOI 10.17487/RFC7223, May 2014,
<<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for
Constrained-Node Networks", RFC 7228,
DOI 10.17487/RFC7228, May 2014,
<<http://www.rfc-editor.org/info/rfc7228>>.

Appendix A. File "ietf-cool.yang"

Module containing the different definitions required by the CoOL protocol.

```
<CODE BEGINS> file "ietf-cool@2016-01-01.yang"
module ietf-cool {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-cool";
  prefix cool;

  organization
    "IETF Core Working Group";

  contact
    "Ana Minaburo
    <mailto:ana@ackl.io>

    Abhinav Somaraju
    <mailto:abhinav.somaraju@tridonic.com>

    Alexander Pelov
    <mailto:a@ackl.io>

    Michel Veillette
    <mailto:michel.veillette@trilliantinc.com>

    Randy Turner
    <mailto:Randy.Turner@landisgyr.com>";

  description
    "This module contains the different definitions required
    by the CoOL protocol.";

  revision 2016-01-01 {
    description
      "Initial revision.";
    reference
      "draft-veillette-core-cool";
  }

  // List of useful derived YANG data types for constrained devices

  typedef sid {
    type uint32;
    description
      "Structure Identifier value (SID).";
  }
```

```
typedef utc-time {
    type uint32;
    description
        "Unsigned 32-bit value representing the number of seconds
        since 0 hours, 0 minutes, 0 seconds, on the 1st of January,
        2000 UTC (Universal Coordinated Time).";
}

// Error payload

container error-payload {
    presence "Defines the format of an error payload.";
    description
        "Optional payload of a client error (CoAP response 4.xx)
        or server error (CoAP response 5.xx).";

    leaf error-code {
        type enumeration {
            enum error {
                value 1;
                description "Unspecified error.";
            }

            enum malformed {
                value 2;
                description "Malformed CBOR payload.";
            }

            enum invalid {
                value 3;
                description "The value specified in the request can't be
                    apply to the target data node.";
            }

            enum doesNotExist {
                value 4;
                description "The target data node instance specified in
                    the request doesn't exist.";
            }

            enum alreadyExist {
                value 5;
                description "The target data node instance specified in
                    the request already exists.";
            }

            enum readOnly {
                value 6;
            }
        }
    }
}
```

```
        description "Attempt to update a read-only data node.";
    }
}
mandatory true;
description
    "Error code.";
}

leaf error-text {
    type string;
    description "Textual descriptions of the error.";
}
}

// Notification payload

identity facility-type {
    description
        "A facility code is used to specify the type of process that
        is logging the message. Notifications from different facilities
        may be handled differently. Other YANG module may add new
        facility type as needed.";
}

identity os {
    base facility-type;
    description
        "Notification generated by the operating system.";
}

identity protocol-stack {
    base facility-type;
    description
        "Notification generated by one of the layers of the IP protocol stack.";
}

identity security {
    base facility-type;
    description
        "Security related notification.";
}

identity hardware-monitoring {
    base facility-type;
    description
        "Hardware related notification.";
}
```

```
identity application {
  base facility-type;
  description
    "Notification generated by an application process.";
}

container notification-payload {
  presence "Defines the format of a notification payload.";
  description
    "Definition of the payload of a notification transferred using CoOL.";

  leaf _id {
    type instance-identifier;
    mandatory true;
    description
      "Identifier associated to the notification reported.";
  }

  leaf timestamp {
    type utc-time;
    description
      "Event timestamp. Support of this field is optional
      since its not expected that all implementations have
      implement a real time clock and if so, this clock is
      available at all time.";
  }

  leaf sequence-number {
    type uint32;
    description
      "Sequence number associated to each event created by CoOL
      server within a specific event stream.";
  }

  leaf severity-level {
    type enumeration {
      enum emergency {
        value 0;
        description
          "System is unusable.";
      }
      enum alert {
        value 1;
        description
          "Should be corrected immediately.";
      }
      enum critical {
        value 2;
      }
    }
  }
}
```



```
        description
            "Critical conditions.";
    }
    enum error {
        value 3;
        description
            "Error conditions.";
    }
    enum warning {
        value 4;
        description
            "May indicate that an error will occur if action is
            not taken.";
    }
    enum notice {
        value 5;
        description
            "Events that are unusual, but not error conditions.";
    }
    enum informational {
        value 6;
        description
            "Normal operational messages that require no action.";
    }
    enum debug {
        value 7;
        description
            "Information useful to developers for debugging the
            application.";
    }
}
description
    "Severity associated with this event.";
reference "RFC 5424";
}

leaf facility {
    type identityref {
        base facility-type;
    }
    description
        "Type of process that is logging the message.";
    reference "RFC 5424";
}

anydata content {
    description
        "Notification container as defined by the notification YANG
```

```
        statement.";
    }
}

rpc commit {
    description
        "Used to commit the changes present in a candidate datastore on
        the runtime datastore specify by the URI used to execute this
        operation.";
    input {
        leaf datastore {
            type string;
            description
                "Path of the datastore resource used as the source of the
                commit operation. When not present, the default candidate
                datastore resource is used.";
        }

        leaf commit-date-time {
            type utc-time;
            description
                "When specified, the commit operation is postponed at the
                specified date and time. When not present, the commit is
                performed on reception of this RPC. Supports of this feature
                is optional.";
        }

        leaf confirm-timeout {
            type string;
            description
                "When present, a confirming commit MUST be received within
                this period after the start of the commit process.
                A confirming commit is a commit RPC without the
                confirm-timeout field presents. Supports of this feature
                is optional.";
        }
    }
}

rpc cancel-commit {
    description
        "Cancels an ongoing scheduled or confirmed commit.";
}
}
<CODE ENDS>
```

Appendix B. File "ietf-cool@2016-01-01.sid"

Following is the ".sid" file generated for the "ietf-cool" YANG module. See [I-D.somaraju-core-sid] for more details on SID and ".sid" file.

```
{
  "assignment-ranges": [
    {
      "entry-point": 1000,
      "size": 100
    }
  ],
  "module-name": "ietf-cool",
  "module-revision": "2016-01-01",
  "items": [
    {
      "type": "Module",
      "label": "ietf-cool",
      "sid": 1000
    },
    {
      "type": "identity",
      "label": "/facility-type",
      "sid": 1001
    },
    {
      "type": "identity",
      "label": "/facility-type/application",
      "sid": 1002
    },
    {
      "type": "identity",
      "label": "/facility-type/hardware-monitoring",
      "sid": 1003
    },
    {
      "type": "identity",
      "label": "/facility-type/os",
      "sid": 1004
    },
    {
      "type": "identity",
      "label": "/facility-type/protocol-stack",
      "sid": 1005
    },
    {
      "type": "identity",
```

```
    "label": "/facility-type/security",
    "sid": 1006
  },
  {
    "type": "node",
    "label": "/error-payload",
    "sid": 1007
  },
  {
    "type": "node",
    "label": "/error-payload/error-code",
    "sid": 1008
  },
  {
    "type": "node",
    "label": "/error-payload/error-text",
    "sid": 1009
  },
  {
    "type": "node",
    "label": "/notification-payload",
    "sid": 1010
  },
  {
    "type": "node",
    "label": "/notification-payload/_id",
    "sid": 1011
  },
  {
    "type": "node",
    "label": "/notification-payload/content",
    "sid": 1012
  },
  {
    "type": "node",
    "label": "/notification-payload/facility",
    "sid": 1013
  },
  {
    "type": "node",
    "label": "/notification-payload/sequence-number",
    "sid": 1014
  },
  {
    "type": "node",
    "label": "/notification-payload/severity-level",
    "sid": 1015
  },
}
```

```
{
  "type": "node",
  "label": "/notification-payload/timestamp",
  "sid": 1016
},
{
  "type": "rpc",
  "label": "/cancel-commit",
  "sid": 1017
},
{
  "type": "rpc",
  "label": "/commit",
  "sid": 1018
},
{
  "type": "rpc",
  "label": "/commit/input/commit-date-time",
  "sid": 1019
},
{
  "type": "rpc",
  "label": "/commit/input/confirm-timeout",
  "sid": 1020
},
{
  "type": "rpc",
  "label": "/commit/input/datastore",
  "sid": 1021
}
]
```

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov (editor)
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: <http://www.landisgyr.com/>

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: September 12, 2016

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
A. Somaraju
Tridonic GmbH & Co KG
R. Turner
Landis+Gyr
A. Minaburo
Acklio
March 11, 2016

CBOR Encoding of Data Modeled with YANG
draft-veillette-core-yang-cbor-mapping-00

Abstract

This document defines encoding rules for serializing configuration data, state data, RPC input and RPC output, Action input, Action output and notifications defined within YANG modules using the Concise Binary Object Representation (CBOR) [RFC7049].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology and Notation	3
2.1. CBOR diagnostic notation	4
3. Properties of the CBOR Encoding	5
4. Encoding of YANG Schema Node Instances	6
4.1. The "leaf" Schema Node	6
4.2. The "container" Schema Node	6
4.3. The "leaf-list" Schema Node	9
4.4. The "list" Schema Node	10
4.5. The "anydata" Schema Node	16
4.6. The "anyxml" Schema Node	17
5. Representing YANG Data Types in CBOR	17
5.1. The unsigned integer Types	17
5.2. The integer Types	17
5.3. The "decimal64" Type	18
5.4. The "string" Type	18
5.5. The "boolean" Type	19
5.6. The "enumeration" Type	19
5.7. The "bits" Type	19
5.8. The "binary" Type	20
5.9. The "leafref" Type	20
5.10. The "identityref" Type	21
5.11. The "empty" Type	22
5.12. The "union" Type	23
5.13. The "instance-identifier" Type	23
6. Security Considerations	28
7. Acknowledgments	28
8. References	29
8.1. Normative References	29
8.2. Informative References	29
Authors' Addresses	30

1. Introduction

The specification of the YANG 1.1 data modelling language [I-D.ietf-netmod-rfc6020bis] defines only an XML encoding for data instances, i.e. contents of configuration datastores, state data, RPC inputs and outputs, action inputs and outputs, and event notifications.

A new set of encoding rules has been defined to allow the use of the same data models in environments based on the JavaScript Object Notation (JSON) Data Interchange Format [RFC7159]. This is accomplished in the JSON Encoding of Data Modeled with YANG specification [I-D.ietf-netmod-yang-json].

The aim of this document is to define a set of encoding rules for the Concise Binary Object Representation (CBOR) [RFC7049]. The resulting encoding is more compact compared to XML and JSON and more suitable for Constrained Nodes and/or Constrained Networks as defined by [RFC7228].

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in [I-D.ietf-netmod-rfc6020bis]:

- o action
- o anydata
- o anyxml
- o data node
- o data tree
- o module
- o notification
- o RPC
- o schema node
- o schema tree
- o submodule

The following terms are defined in [I-D.ietf-netmod-yang-json]:

- o member name
- o name of an identity

The following term is defined in [I-D.vanderstok-core-comi]:

- o YANG hash

This specification also makes use of the following terminology:

- o child: A schema node defined within a collection such as a container, a list, a case, a notification, an RPC input, an RPC output, an action input, an action output.
- o delta : Difference between the SID assigned to the current schema node and the SID assigned to the parent.
- o parent: The collection in which a schema node is defined.
- o structured identifier or SID: Unsigned integer used to identify different YANG items.

2.1. CBOR diagnostic notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation as defined in [RFC7049] section 6. This notation is used strictly for documentation purposes and is never used in the data serialization.

CBOR content	CBOR type	Diagnostic notation	Example	CBOR encoding
Unsigned integer	0	Decimal digits	123	18 7b
Negative integer	1	Decimal digits prefixed by a minus sign	-123	38 7a
Byte string	2	Hexadecimal value enclosed between single quotes and prefixed by an 'h'	h'f15c'	42 f15c
Text string	3	String of Unicode characters enclosed between double quotes	"txt"	63 747874
Array	4	Comma-separated list of values within square brackets	[1, 2]	82 01 02
Map	5	Comma-separated list of key : value pairs within curly braces	{ 1: 123, 2: 456 }	a2 01187b 021901c8
Boolean	7/20 7/21	false true	false true	f4 f5
Null	7/22	null	null	f6
Not assigned	7/23	undefined	undefined	f7

Within this document, comments are allowed in CBOR diagnostic notation. Any characters after a Pound sign ('#') outside of a string, up to the end of the line, are treated as a comment.

3. Properties of the CBOR Encoding

This document defines CBOR encoding rules for YANG schema trees and their subtrees.

Basic schema nodes such as leaf, leaf-list, list, anydata and anyxml can be encoded standalone. In this case, only the value of this schema node is encoded in CBOR. Identification of this value need to be provided by some external means when needed.

A collection such as container, list instance, notification, RPC input, RPC output, action input and action output is serialized using a CBOR map in which each child schema node is encoded using a key and a value. This specification supports three type of keys; SID as defined in [I-D.somaraju-core-sid], member names as defined in [I-D.ietf-netmod-yang-json] and YANG hash as defined by

[I-D.vanderstok-core-comi]. Each of these key type is encoded using a specific CBOR type which allows their interpretation during the deserialization process. The end user of this mapping specification can mandate the use of a specific key type or a specific subset of key types.

In order to minimize the size of the encoded data, the proposed mapping does not make use of any meta-information beyond those natively supported by CBOR. For instance, CBOR tags are not used for any of the proposed mapping. It is expected that entities generating and decoding CBOR contents have enough knowledge about the information processed in order to perform the expected task without the need of such extra meta-information. The CoAP Content-Format Option, or an HTTP Content-Type header field, conveys that the data is YANG-encoded CBOR in the first place.

4. Encoding of YANG Schema Node Instances

Schema node instances defined using the YANG modeling language are encoded using CBOR [RFC7049] based on the rules defined in this section. We assume that the reader is already familiar with both YANG [I-D.ietf-netmod-rfc6020bis] and CBOR [RFC7049].

4.1. The "leaf" Schema Node

Leafs MUST be encoded based on the encoding rules specified in Section 5.

4.2. The "container" Schema Node

Collections such as containers, list instances, notifications, RPC inputs, RPC outputs, action inputs and action outputs MUST be encoded using a CBOR map data item (major type 5). A map is comprised of pairs of data items, with each data item consisting of a key and a value. This specification supports three type of keys; SID as defined in [I-D.somaraju-core-sid], member names as defined in [I-D.ietf-netmod-yang-json] and YANG hash as defined by [I-D.vanderstok-core-comi].

SIDs as keys

Keys implemented using SIDs MUST be encoded using a CBOR unsigned integer (major type 0) and set to the delta value of the associated SID. Delta values are computed as follows:

- o The delta value is equal to the SID of the current schema node minus the SID of the parent schema node. When no parent exists in the context of use of this container, the delta is set to the SID

of the current schema node (a parent with SID equal to zero is assumed).

- o Delta values may result in a negative number, clients and servers MUST support negative deltas.

Member names as keys

Keys implemented using member names MUST be encoded using a CBOR text string data item (major type 3). A namespace-qualified member name MUST be used for all members of a top-level collection, and then also whenever the namespaces of the schema node and its parent are different. In all other cases, the simple form of the member name MUST be used. Member names and namespaces are defined in [I-D.ietf-netmod-yang-json] section 4.

YANG hashes as keys

Keys implemented using YANG hashes MUST be encoded using a CBOR byte string data item (major type 2).

Values MUST be encoded using the appropriate rules defined in Section 4 and Section 5.

Definition example [RFC7317]:

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
      \d{2}:\d{2})';
  }
}

container system {
  leaf hostname {
    type inet:domain-name;

    container clock {
      leaf current-datetime {
        type date-and-time;
      }

      leaf boot-datetime {
        type date-and-time;
      }
    }
  }
}
```

SIDs example

This example is encoded using the SIDs defined in [I-D.somaraju-core-sid] Appendix C.

CBOR diagnostic notation:

```
{
  1708 : {
    2 : "2015-10-02T14:47:24Z-05:00", # current-datetime, SID 1710
    1 : "2015-09-15T09:12:58Z-05:00" # boot-datetime, SID 1709
  }
}
```

CBOR encoding:

```
a1 # map(1)
  19 06ac # unsigned(1708)
  a2 # map(2)
    02 # unsigned(2)
    78 1a # text(26)
    323031352d31302d30325431343a34373a32345a2d30353a3030
    01 # unsigned(1)
    78 1a # text(26)
    323031352d30392d31355430393a31323a35385a2d30353a3030
```

Member names example

CBOR diagnostic notation:

```
{
  "ietf-system:clock" : {
    "current-datetime" : "2015-10-02T14:47:24Z-05:00",
    "boot-datetime" : "2015-09-15T09:12:58Z-05:00"
  }
}
```

CBOR encoding:

```

a1                                     # map(1)
  71                                 # text(17)
    696574662d73797374656d3a636c6f636b # "ietf-system:clock"
  a2                                 # map(2)
    70                              # text(16)
      63757272656e742d6461746574696d65 # "current-datetime"
    78 1a                           # text(26)
      323031352d31302d30325431343a34373a32345a2d30353a3030
    6d                              # text(13)
      626f66742d6461746574696d65 # "boot-datetime"
    78 1a                           # text(26)
      323031352d30392d31355430393a31323a35385a2d30353a3030

```

YANG Hashes example

CBOR diagnostic notation:

```

{
  h'334c67d9' : {                  # clock
    h'047c468b' : "2015-10-02T14:47:24Z-05:00", # current-datetime
    h'2fe1a167' : "2015-09-15T09:12:58Z-05:00"  # boot-datetime
  }
}

```

CBOR encoding:

```

a1                                     # map(1)
  44                                 # bytes(4)
    334c67d9
  a2                                 # map(2)
    44                              # bytes(4)
      047c468b
    78 1a                           # text(26)
      323031352d31302d30325431343a34373a32345a2d30353a3030
    44                              # bytes(4)
      2fe1a167
    78 1a                           # text(26)
      323031352d30392d31355430393a31323a35385a2d30353a3030

```

4.3. The "leaf-list" Schema Node

A leaf-list MUST be encoded using a CBOR array data item (major type 4). Each entry of this array MUST be encoded using the rules defined by the YANG type specified.

Definition example [RFC7317]:

```
typedef domain-name {  
    type string {  
        length "1..253";  
        pattern '((([a-zA-Z0-9_]([a-zA-Z0-9\_-]){0,61})?[a-zA-Z0-9].)  
                *([a-zA-Z0-9_]([a-zA-Z0-9\_-]){0,61})?[a-zA-Z0-9]\.?  
                )|\.';  
    }  
}  
  
leaf-list search {  
    type domain-name;  
    ordered-by user;  
}
```

CBOR diagnostic notation: ["ietf.org", "ieee.org"]

CBOR encoding: 82 68 696574662e6f7267 68 696565652e6f7267

4.4. The "list" Schema Node

A list MUST be encoded using a CBOR array data item (major type 4). Each entry of this array is encoded using a CBOR map data item (major type 5) based on the same rules as a YANG container, see Section 4.2.

Definition example [RFC7317]:


```
list server {
  key name;

  leaf name {
    type string;
  }
  choice transport {
    case udp {
      container udp {
        leaf address {
          type host;
          mandatory true;
        }
        leaf port {
          type port-number;
        }
      }
    }
  }
  leaf association-type {
    type enumeration {
      enum server;
      enum peer;
      enum pool;
    }
    default server;
  }
  leaf iburst {
    type boolean;
    default false;
  }
  leaf prefer {
    type boolean;
    default false;
  }
}
```

SIDs example

SIDs used in this example are defined in [I-D.somaraju-core-sid] Appendix C. It is important to note that the protocol or method using this mapping may carry a parent SID or may have the knowledge of this parent SID based on its context. In these cases, delta encoding can be performed based on this parent SID which minimizes the size of the encoded data.

CBOR diagnostic notation:

```
[
  {
    1746 : "NRC TIC server",      # name
    1748 : {                      # udp
      1 : "tic.nrc.ca",          # address, SID 1749
      2 : 123                    # port, SID 1750
    },
    1744 : 0,                    # association-type
    1745 : false,                 # iburst
    1747 : true                   # prefer
  },
  {
    1746 : "NRC TAC server",      # name
    1748 : {                      # udp
      1 : "tac.nrc.ca"           # address, SID 1749
    }
  }
]
```

CBOR encoding:

```
82                                # array(2)
a5                                # map(5)
  19 06d2                         # unsigned(1746)
  6e                               # text(14)
    4e52432054494320736572766572 # "NRC TIC server"
  19 06d4                         # unsigned(1748)
  a2                               # map(2)
    01                            # unsigned(1)
    6a                            # text(10)
      74696332e6e72632e6361      # "tic.nrc.ca"
    02                            # unsigned(2)
    18 7b                         # unsigned(123)
  19 06d0                         # unsigned(1744)
  00                              # unsigned(0)
  19 06d1                         # unsigned(1745)
  f4                              # primitive(20)
  19 06d3                         # unsigned(1747)
  f5                              # primitive(21)
a2                                # map(2)
  19 06d2                         # unsigned(1746)
  6e                               # text(14)
    4e52432054414320736572766572 # "NRC TAC server"
  19 06d4                         # unsigned(1748)
  a1                               # map(1)
    01                            # unsigned(1)
    6a                            # text(10)
      74616332e6e72632e6361      # "tac.nrc.ca"
```

Member names example

CBOR diagnostic notation:

```
[
  {
    "ietf-system:name" : "NRC TIC server",
    "ietf-system:udp" : {
      "address" : "tic.nrc.ca",
      "port" : 123
    },
    "ietf-system:association-type" : 0,
    "ietf-system:iburst" : false,
    "ietf-system:prefer" : true
  },
  {
    "ietf-system:name" : "NRC TAC server",
    "ietf-system:udp" : {
      "address" : "tac.nrc.ca"
    }
  }
]
```

CBOR encoding:

```

82                                     # array(2)
  a5                                 # map(5)
    70                               # text(16)
      696574662d73797374656d3a6e616d65 # "ietf-system:name"
    6e                               # text(14)
      4e52432054494320736572766572      # "NRC TIC server"
    6f                               # text(15)
      696574662d73797374656d3a756470      # "ietf-system:udp"
    a2                               # map(2)
      67                               # text(7)
        61646472657373                  # "address"
      6a                               # text(10)
        74696332e6e72632e6361          # "tic.nrc.ca"
      64                               # text(4)
        706f7274                        # "port"
    18 7b                             # unsigned(123)
    78 1c                             # text(28)
      696574662d73797374656d3a6173736f6369617469666e2d74797065
    00                               # unsigned(0)
    72                               # text(18)
      696574662d73797374656d3a696275727374 # "ietf-system:iburst"
    f4                               # primitive(20)
    72                               # text(18)
      696574662d73797374656d3a707265666572 # "ietf-system:prefer"
    f5                               # primitive(21)
  a2                               # map(2)
    70                               # text(16)
      696574662d73797374656d3a6e616d65 # "ietf-system:name"
    6e                               # text(14)
      4e52432054414320736572766572      # "NRC TAC server"
    6f                               # text(15)
      696574662d73797374656d3a756470      # "ietf-system:udp"
    a1                               # map(1)
      67                               # text(7)
        61646472657373                  # "address"
      6a                               # text(10)
        74616332e6e72632e6361          # "tac.nrc.ca"

```

YANG hashes example

CBOR diagnostic notation:

```
[
  {
    h'06c32032' : "NRC TIC server",      # name
    h'11889c84' : {                      # udp
      h'3158c529' : "tic.nrc.ca",        # address
      h'34492d05' : 123                  # port
    },
    h'2c2c2ccf' : 0,                    # association-type
    h'1058dc5d' : false,                 # iburst
    h'390e346a' : true                   # prefer
  },
  {
    h'06c32032' : "NRC TAC server",      # name
    h'11889c84' : {                      # udp
      h'3158c529' : "tac.nrc.ca"        # address
    }
  }
]
```

CBOR encoding:

```

82                                     # array(2)
  a5                                 # map(5)
    44                               # bytes(4)
      06c32032"
    6e                               # text(14)
      4e52432054494320736572766572 # "NRC TIC server"
    44                               # bytes(4)
      11889c84 "
  a2                                 # map(2)
    44                               # bytes(4)
      3158c529
    6a                               # text(10)
      7469632e6e72632e6361          # "tic.nrc.ca"
    44                               # bytes(4)
      34492d05
    18 7b                           # unsigned(123)
    44                               # bytes(4)
      2c2c2ccf
    00                               # unsigned(0)
    44                               # bytes(4)
      1058dc5d
    f4                               # primitive(20)
    44                               # bytes(4)
      390e346a
  f5                               # primitive(21)
  a2                                 # map(2)
    44                               # bytes(4)
      06c32032
    6e                               # text(14)
      4e52432054414320736572766572 # "NRC TAC server"
    44                               # bytes(4)
      11889c84
    a1                               # map(1)
      44                               # bytes(4)
        3158c529
      6a                               # text(10)
        7461632e6e72632e6361        # "tac.nrc.ca"

```

4.5. The "anydata" Schema Node

An anydata serves as a container for an arbitrary set of schema nodes that otherwise appear as normal YANG-modeled data. An anydata instance is encoded using the same rules as a container, i.e., CBOR map. The requirement that anydata content can be modeled by YANG implies the following:

- o Keys MUST be set to valid SIDs, member names or YANG hashes. This rule applies to the key of the anydata node and the key of any inner schema node.
- o The CBOR array MUST contain either unique scalar values (as a leaf-list, see Section 4.3), or maps (as a list, see Section 4.4).
- o Values MUST follow the encoding rules of one of the datatypes listed in Section 5.

4.6. The "anyxml" Schema Node

An anyxml instance is encoded as a CBOR key/value pair. The key of the anyxml schema node MUST be a valid SID, member name or YANG hash but the value is unrestricted, i.e., the value can be any CBOR encoded content.

5. Representing YANG Data Types in CBOR

5.1. The unsigned integer Types

Leafs of type uint8, uint16, uint32 and uint64 MUST be encoded using a CBOR unsigned integer data item (major type 0).

Definition example [RFC7277]:

```
leaf mtu {  
  type uint16 {  
    range "68..max";  
  }  
}
```

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

5.2. The integer Types

Leafs of type int8, int16, int32 and int64 MUST be encoded using either CBOR unsigned integer (major type 0) or CBOR signed integer (major type 1), depending on the actual value.

Definition example [RFC7317]:

```
leaf timezone-utc-offset {  
  type int16 {  
    range "-1500 .. 1500";  
  }  
}
```

CBOR diagnostic notation: -300

CBOR encoding: 39 012b

5.3. The "decimal64" Type

Leafs of type decimal64 MUST be encoded using either CBOR unsigned integer (major type 0) or CBOR signed integer (major type 1), depending on the actual value. The position of the decimal point is defined by the fraction-digits YANG statement and is not available in the CBOR encoding.

Definition example [RFC7317]:

```
leaf my-decimal {  
  type decimal64 {  
    fraction-digits 2;  
    range "1 .. 3.14 | 10 | 20..max";  
  }  
}
```

CBOR diagnostic notation: 257 (Represents decimal value 2.57)

CBOR encoding: 19 0101

5.4. The "string" Type

Leafs of type string MUST be encoded using a CBOR text string data item (major type 3).

Definition example [RFC7223]:

```
leaf name {  
  type string;  
}
```

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

5.5. The "boolean" Type

Leafs of type boolean MUST be encoded using a CBOR true (major type 7, additional information 21) or false data item (major type 7, additional information 20).

Definition example [RFC7317]:

```
leaf enabled {  
    type boolean;  
}
```

CBOR diagnostic notation: true

CBOR encoding: f5

5.6. The "enumeration" Type

Leafs of type enumeration MUST be encoded using a CBOR unsigned integer data item (major type 0).

Definition example [RFC7317]:

```
leaf oper-status {  
    type enumeration {  
        enum up { value 1; }  
        enum down { value 2; }  
        enum testing { value 3; }  
        enum unknown { value 4; }  
        enum dormant { value 5; }  
        enum not-present { value 6; }  
        enum lower-layer-down { value 7; }  
    }  
}
```

CBOR diagnostic notation: 3 (Represents enumeration value "testing")

CBOR encoding: 03

5.7. The "bits" Type

Leafs of type bits MUST be encoded using a CBOR byte string data item (major type 2). Bits position 0 to 7 are assigned to the first byte within the byte string, bits 8 to 15 to the second byte, and subsequent bytes are assigned similarly. Within each byte, bits are assigned from least to most significant.

Definition example [I-D.ietf-netmod-rfc6020bis]:

```
leaf mybits {  
  type bits {  
    bit disable-nagle {  
      position 0;  
    }  
    bit auto-sense-speed {  
      position 1;  
    }  
    bit 10-Mb-only {  
      position 2;  
    }  
  }  
}
```

CBOR diagnostic notation: h'05' (Represents bits disable-nagle and 10-Mb-only set)

CBOR encoding: 41 05

5.8. The "binary" Type

Leafs of type binary MUST be encoded using a CBOR byte string data item (major type 2).

Definition example:

```
leaf aes128-key {  
  type binary {  
    length 16;  
  }  
}
```

CBOR diagnostic notation: h'1f1ce6a3f42660d888d92a4d8030476e'

CBOR encoding: 50 1f1ce6a3f42660d888d92a4d8030476e

5.9. The "leafref" Type

Leafs of type leafref MUST be encoded using the rules of the schema node referenced by the "path" YANG statement.

Definition example [RFC7223]:

```
typedef interface-state-ref {
  type leafref {
    path "/interfaces-state/interface/name";
  }
}

container interfaces-state {
  list interface {
    key "name";
    leaf name {
      type string;
    }
    leaf-list higher-layer-if {
      type interface-state-ref;
    }
  }
}
```

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

5.10. The "identityref" Type

This specification supports two approaches for encoding `identityref`, a SID as defined in [I-D.somaraju-core-sid] or a name as defined in [I-D.ietf-netmod-yang-json] section 6.8.

SIDs as identityref

SIDs are globally unique and may be used as `identityref`. This approach is both compact and simple to implement. When SIDs are used, `identityref` MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to a SID allocated from a registered SID range.

Name as identityref

Alternatively, an `identityref` may be encoded using a name as defined in [I-D.ietf-netmod-yang-json] section 6.8. When names are used, `identityref` MUST be encoded using a CBOR text string data item (major type 3). If the identity is defined in another module than the leaf node containing the `identityref` value, the namespace-qualified form MUST be used. Otherwise, both the simple and namespace-qualified forms are permitted.

Definition example [RFC7223]:

```
identity interface-type {  
}  
  
identity iana-interface-type {  
  base interface-type;  
}  
  
identity ethernetCsmacd {  
  base iana-interface-type;  
}  
  
leaf type {  
  type identityref {  
    base interface-type;  
  }  
}
```

SIDs as identityref

Assuming that the identity "iana-if-type:ethernetCsmacd" has been assigned to the SID value 1179.

CBOR diagnostic notation: 1179

CBOR encoding: 19 049b

Name as identityref

CBOR diagnostic notation: "iana-if-type:ethernetCsmacd"

CBOR encoding: 78 1b

69616e612d696662d747970653a65746865726e657443736d616364

5.11. The "empty" Type

Leafs of type empty MUST be encoded using the CBOR null value (major type 7, additional information 22).

Definition example [RFC7277]:

```
leaf is-router {  
  type empty;  
}
```

CBOR diagnostic notation: null

CBOR encoding: f6

5.12. The "union" Type

Leafs of type union MUST be encoded using the rules associated with one of the types listed.

Definition example [RFC7317]:

```
typedef ipv4-address {
    type string {
        pattern '((( [0-9] | [1-9] [0-9] | 1 [0-9] [0-9] | 2 [0-4] [0-9] | 25 [0-5] ) \. ) {3}
                ([0-9] [1-9] [0-9] | 1 [0-9] [0-9] | 2 [0-4] [0-9] | 25 [0-5] ) ( % [ \p{N}
                \p{L} ] + ) ? ' ;
    }
}

typedef ipv6-address {
    type string {
        pattern '((( (: | [0-9a-fA-F] {0,4} ) : ) ([0-9a-fA-F] {0,4} : ) {0,5} ((( [0-9a-
        fA-F] {0,4} : ) ? ( : | [0-9a-fA-F] {0,4} ) ) | ( ( (25 [0-5] | 2 [0-4] [0-
        9] | [01] ? [0-9] ? [0-9] ) \. ) {3} (25 [0-5] | 2 [0-4] [0-9] | [01] ? [0-
        9] ? [0-9] ) ) ) ( % [ \p{N} \p{L} ] + ) ? ' ;
        pattern '((( [^: ] + : ) {6} ([^: ] + : [^: ] + ) | ( . * \. . * ) ) | ( ( ( [^: ] + : ) * [^: ] + )
        ? : : ( ( [^: ] + : ) * [^: ] + ) ? ) ( % . + ) ? ' ;
    }
}

typedef ip-address {
    type union {
        type ipv4-address;
        type ipv6-address;
    }
}

leaf address {
    type inet:ip-address;
}

```

CBOR diagnostic notation: "2001:db8:a0b:12f0::1"

CBOR encoding: 74 323030313a6462383a6130623a313266303a3a31

5.13. The "instance-identifier" Type

This specification supports three approaches for encoding an instance-identifier, one based on SIDs as defined in [I-D.somaraju-core-sid], one based on names as defined in [I-D.ietf-netmod-yang-json] section 6.13 and one based on YANG hashes as defined in [I-D.vanderstok-core-comil].

SIDs as instance-identifier

SIDs uniquely identify a data node. For a single instance data node, the SID is sufficient to identify this instance. For a multi-instance data node, a SID is combined with the list key(s) to identify each instance of this data node within the YANG list(s).

Single instance data nodes MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.

Multi-instances data nodes MUST be encoded using a CBOR array data item (major type 4) containing the following entries:

- o The first entry MUST be encoded as a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.
- o The following entries MUST contain the value of each key required to identify the instance of the targeted data node. These keys MUST be ordered as defined in the "key" YANG statement, starting from top level list, and follow by each of the subordinate list(s).

When SIDs identify a YANG list, the presence of the key(s) for this list is optional. When the key(s) are present, the targeted instance within this list is selected. When the key(s) are absent, the entire YANG list is selected.

Names as instance-identifier

The use of names as instance-identifier is defined in [I-D.ietf-netmod-yang-json] section 6.11. The resulting xpath MUST be encoded using a CBOR text string data item (major type 3).

YANG hashes as instance-identifier

When YANG hashes are used, xpath can be compressed based on the method defined by [I-D.vanderstok-core-comi] sections 4.1.4.1 and 4.1.4.2.

Definition example [RFC7317]:

```
container system {  
    leaf contact {  
        type string;  
    }  
  
    leaf hostname {  
        type inet:domain-name;  
    }  
}
```

First example based on SID

In this example, a field of type instance-identifier identifies the data node `"/system/contact"` (SID 1728).

1728

CBOR encoding:

19 06c0

First example based on name

Same example as above based on names.

`"/ietf-system:system/contact"`

CBOR encoding:

78 1c 2f20696574662d73797374656d3a73797374656d2f636f6e74616374

First example based on YANG hash

Same example assuming data node `"/system/contact"` is associated to YANG hash `0x09b06d17` or `"JsG0X"` in base64.

CBOR diagnostic notation:

`"/JsG0X"`

CBOR encoding:

66 2f4a73473058

Definition example [RFC7317]:

```
list user {
  key name;

  leaf name {
    type string;
  }
  leaf password {
    type ianach:crypt-hash;
  }
}

list authorized-key {
  key name;

  leaf name {
    type string;
  }
  leaf algorithm {
    type string;
  }
  leaf key-data {
    type binary;
  }
}
```

Second example based on SID

In this example, a field of type instance-identifier identify the data node `"/system/authentication/user/authorized-key/key-data"` (SID 1721) for the user name `"bob"` and the authorized-key name `"admin"`.

CBOR diagnostic notation:

```
[1721, "bob", "admin"]
```

CBOR encoding:

83		# array(3)
19	06b9	# unsigned(1721)
63		# text(3)
	626f62	# "bob"
65		# text(5)
	61646d696e	# "admin"

Second example based on name

Same example as above based on names.

CBOR diagnostic notation:


```
"/ietf-system:system/authentication/user[name='bob']/authorized-key
[name='admin']/key-data"
```

CBOR encoding:

```
78 59
  2f696574662d73797374656d3a73797374656d2f61757468656e74696361
  746966f6e2f757365725b6e616d653d27626f62275d2f617574686f72697a
  65642d6b65795b6e616d653d2761646d696e275d2f6b65792d64617461
```

Second example based on YANG hash

Same example assuming data node `"/ietf-system:system/authentication/user/authorized-key/key-data"` is associated to YANG hash `0x0d6e7afb` or `"Nbnr7"` in base64.

CBOR diagnostic notation:

```
"/Nbnr7?keys=\"bob\", \"admin\""
```

CBOR encoding:

```
78 19 2f4e626e72373f6b6579733d22626f62222c2261646d696e22
```

Third example based on SID

This third example identify an instance within the list `"/system/authentication/user"` (SID 1717) corresponding to the user name `"jack"`.

CBOR diagnostic notation:

```
[1717, "jack"]
```

CBOR encoding:

```
82                                # array(2)
  19 06b5                        # unsigned(1717)
  64                              # text(4)
    6a61636b                     # "jack"
```

Third example based on name

Same example as above based on names.

CBOR diagnostic notation:

```
"/ietf-system:system/authentication/user[name='bob']"
```

CBOR encoding:

```
78 33
 2f696574662d73797374656d3a73797374656d2f61757468656e74696361
 746966f6e2f757365725b6e616d653d27626f62275d
```

Third example based on YANG hash

Same example assuming data node `"/ietf-system:system/authentication/user"` is associated to YANG hash `0x2677c6c1` or `"md8bB"` in base64.

CBOR diagnostic notation:

```
"/md8bB?keys=\"bob\""
```

CBOR encoding:

```
71 2f6d643862423f6b6579733d22626f6222
```

6. Security Considerations

The security considerations of [RFC7049] and [I-D.ietf-netmod-rfc6020bis] apply.

This document defines an alternative encoding for data modeled in the YANG data modeling language. As such, this encoding does not contribute any new security issues in addition of those identified for the specific protocol or context for which it is used.

To minimize security risks, software on the receiving side SHOULD reject all messages that do not comply to the rules of this document and reply with an appropriate error message to the sender.

7. Acknowledgments

This document has been largely inspired by the extensive works done by Andy Bierman and Peter van der Stok on [I-D.vanderstok-core-comi]. [I-D.ietf-netmod-yang-json] has also been a critical input to this work. The authors would like to thank the authors and contributors to these two drafts.

The authors would also like to acknowledge the review, feedback, and comments from Ladislav Lhotka and Juergen Schoenwaelder.

8. References

8.1. Normative References

- [I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language",
draft-ietf-netmod-rfc6020bis-11 (work in progress),
February 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.

8.2. Informative References

- [I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG",
draft-ietf-netmod-yang-json-09 (work in progress), March
2016.
- [I-D.vanderstok-core-comi]
Stok, P. and A. Bierman, "CoAP Management Interface",
draft-vanderstok-core-comi-09 (work in progress), March
2016.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March
2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface
Management", RFC 7223, DOI 10.17487/RFC7223, May 2014,
<<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for
Constrained-Node Networks", RFC 7228,
DOI 10.17487/RFC7228, May 2014,
<<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management",
RFC 7277, DOI 10.17487/RFC7277, June 2014,
<<http://www.rfc-editor.org/info/rfc7277>>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov (editor)
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: <http://www.landisgyr.com/>

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: March 31, 2016

T. Zotti
Philips Research
P. van der Stok
Consultant
E. Dijk
Philips Research
September 28, 2015

Sleepy CoAP Nodes
draft-zotti-core-sleepy-nodes-04

Abstract

Control networks rely on application protocols like CoAP to enable RESTful communications in constrained environments. Many of these networks make use of "Sleepy Nodes": battery powered devices that switch off their (radio) interface during most of the time to conserve battery energy. As a result of this, Sleepy Nodes cannot be reached most of the time. This fact prevents using normal communication patterns as specified in the CoRE group, since the server-model is not applicable to these devices. This document discusses and specifies an architecture to support Sleepy Nodes such as battery-powered sensors in mesh networks with the goal of proposing a standardisation solution for Sleepy Node proxies.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 31, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Problem statement	3
1.2. Assumptions	4
1.3. Requirements Language	4
1.4. Acronyms	5
2. Use cases and architecture	5
2.1. Node interactions and use cases	6
2.2. Architecture	9
2.3. Example contents	10
3. Design motivation	10
4. Interactions involving Resource Directory	10
5. Synchronize interface	12
5.1. Sleepy Node discovers proxy	12
5.2. Registration at a Proxy	12
5.3. De-registration at a Proxy	15
5.4. Initialization of delegated resource	16
5.5. Sleepy Node updates delegated resource at Proxy	17
5.6. Sleepy Node READs resource updates from Proxy	18
6. Delegate Interface	18
6.1. Discovering Endpoint discovers Sleepy Node at Proxy	19
6.2. Proxy REPORTs events to Endpoint	20
6.3. A Node WRITES to Sleepy Node via Proxy	21
6.4. A Node READs information from Sleepy Node via Proxy	22
7. Direct Interface	22
7.1. Sleepy Node REPORTs events directly to Destination Node	22
7.2. A Sleepy Node READs information from a Server Node	23
8. Realization with PubSub broker	23
9. IANA Considerations	23
10. Security Considerations	24
11. Acknowledgements	24
12. Changelog	24
13. References	25
13.1. Normative References	25
13.2. Informative References	25
Authors' Addresses	26

1. Introduction

Control networks rely on application protocols such as CoAP to enable RESTful communications in constrained environments. Many of these networks feature "Sleepy Nodes": battery-powered nodes which switch on/off their communication interface to conserve battery energy. As a result of this, Sleepy Nodes cannot be reached most of the time. This fact prevents using normal communication patterns as specified by the CoRE group, since the server model is clearly not applicable to the most energy constrained devices.

This document discusses and specifies an architecture to support Sleepy Nodes such as battery-powered sensors in wireless networks. The proposed solution makes use of a Proxy Node to which a Sleepy Node delegates part of its communication tasks while it is not accessible in the wireless network. Direct interactions between Sleepy Nodes and non-Sleepy Nodes are only possible, when the Sleepy Node initiates the communication.

Earlier related documents treating the Sleepy Node subject are the CoRE mirror server [I-D.vial-core-mirror-server] and the Publish-Subscribe in the Constrained Application Protocol (CoAP) [I-D.koster-core-coap-pubsub]. Both documents describe the interfaces to the proxy accompanying the Sleepy Node. Both make use of the observe option discussed in [I-D.ietf-core-observe]. This document describes the roles of the nodes communicating with the Sleepy Node and/or its proxy. The draft describes the differences between the concepts supporting the Sleepy Node, and the concepts underlying the PubSub paradigm.

The draft relies heavily on the concepts introduced by the Resource Directory [I-D.ietf-core-resource-directory], and describes how the Sleepy Node profits of the introduction of a Resource Directory into the network.

The issues that need to be addressed to provide support for Sleepy Nodes in Control networks are summarized in Section 1.1. Section 2 provides a set of use case descriptions that introduce communication patterns to be used in home and building control scenarios. Section 4, Section 5, Section 6, and Section 7 specify interfaces to support each of these scenarios. Many interface specifications and examples are taken over from [I-D.vial-core-mirror-server].

1.1. Problem statement

During typical operation, a Sleepy Node has its radio disabled and the CPU may be in a sleeping state. If an external event occurs (e.g. person walks into the room activating a presence sensor), the

CPU and radio are powered back on and they send out a message to another node, or to a group of nodes. After sending this message, the radio and CPU are powered off again, and the Sleepy Node sleeps until the next external event or until a predefined time period has passed. The main problems when introducing Sleepy Nodes into a wireless network are as follows:

Problem 1: How to contact a Sleepy Node that has its radio turned off most of the time for:

- Writing configuration settings.
- Reading out sensor data, settings or log data.
- Configuring additional event destination nodes or node groups.

Problem 2: How to discover a Sleepy Node and its services, while the node is asleep:

- Direct node discovery (CoAP GET /.well-known/core as defined in [RFC7252]) does not find the node with high probability.
- Mechanisms may be needed to provide, as the result of node discovery, the IP address of a Proxy instead of the IP address of the node directly.

Problem 3: How a Sleepy Node can convey data to a node or groups of nodes, with good reliability and minimal energy consumption.

1.2. Assumptions

The solution architecture specified here assumes that a Sleepy Node has enough energy to perform bidirectional communication during its normal operational state. This solution may be applicable also to extreme low-power devices such as solar powered sensors as long as they have enough energy to perform commissioning and the initial registration steps. These installation operations may require, in some cases, an additional source of power. Since a Sleepy Node is unreachable for relatively long periods of times, the data exchanges in the interaction model are always initiated by a Sleepy Node when its sleep period ends.

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document assumes readers are familiar with the terms and concepts discussed in [RFC7252], [RFC5988], [I-D.ietf-core-resource-directory], [I-D.ietf-core-interfaces], [I-D.ietf-core-observe] and [I-D.vial-core-mirror-server].

In addition, this document makes use of the following additional terminology:

Sleepy Node: a battery-powered node which does the on/off switching of its communication interface with the purpose of conserving battery energy

Sleeping/Asleep: A Sleepy Node being in a "sleeping state" i.e. its network interface is switched off and a Sleepy Node is not able to send or receive messages.

Awake/Not Sleeping: A Sleepy Node being in an "awake state" i.e. its network interface is switched on and the Sleepy Node is able to send or receive messages.

Wake up reporting duration: the duration between a wake up from a Sleepy Node and the next wake up and report of the same Node.

Proxy: any node that is configured to, or selected to, perform communication tasks on behalf of one or more Sleepy Nodes.

Regular Node: any node in the network which is not a Proxy or a Sleepy Node.

1.4. Acronyms

This Internet-Draft contains the following acronyms:

DTLS: Datagram Transport Layer Security

EP: Endpoint

MC: Multicast

RD: Resource Directory

2. Use cases and architecture

To describe the application viewpoint of the solution, we introduce some example scenarios for the various interactions shown in Figure 1. The figure assigns the following roles taken up by a regular node:

- o Reading Node: any regular node that reads information from the Sleepy Node.
- o Configuring Node: any regular node that writes information/configuration into Sleepy Node(s). Examples of configuration are new thresholds for a sensor or a new value for the wake-up cycle time.
- o Discovering Node: any regular node that performs discovery of the nodes in a network, including Sleepy Nodes.
- o Destination Node: any regular node or node in a group that receives a message that is generated by the Sleepy Node.
- o Server Node: an optional server that the Sleepy Node knows about, or is told about, which is used to fetch information/configuration/firmware updates/etc.
- o Discovery Server: an optional server that enables nodes to discover all the devices in the network, including Sleepy Nodes, and query their capabilities. For example, a Resource Directory server as defined in [I-D.ietf-core-resource-directory] or a DNS-SD server as defined in [RFC6763]. For the rest of this document the discovery server is a Resource Directory. Specifically, the functionalities of the Resource Directory related to the architecture presented in this Internet-Draft are described in more details in Section 4.
- o Delegated resource is the copy at the Proxy of a resource present in the Sleepy Node.

2.1. Node interactions and use cases

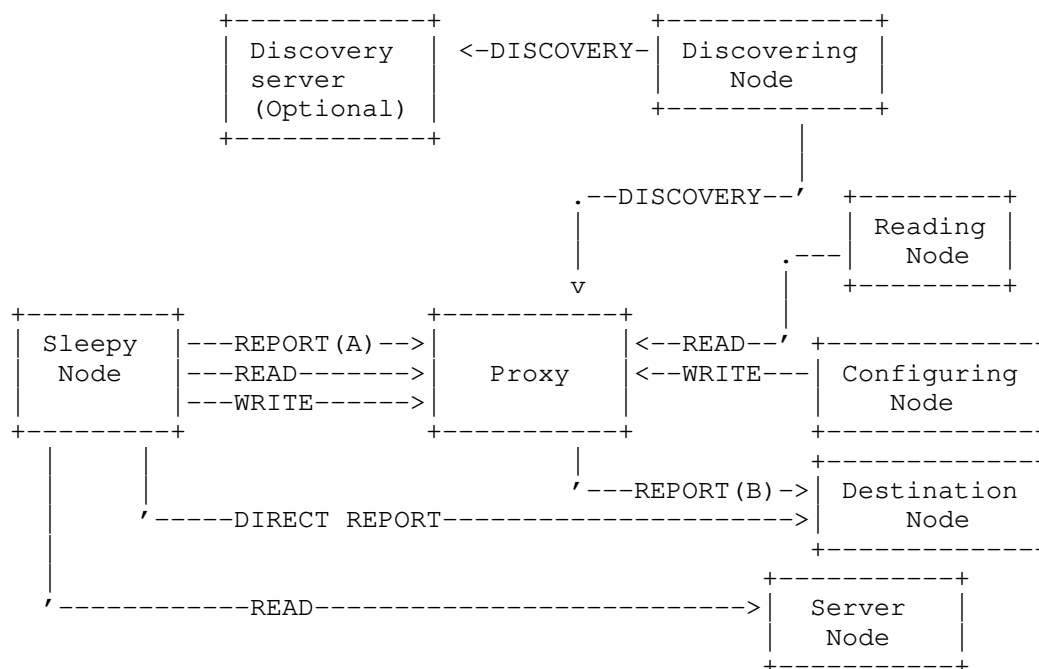


Figure 1: Interaction model for Sleepy Nodes in IP-based networks

The interactions visualized in Figure 1 are discussed and motivated with their use cases. The arrows in the figure indicate that the initiative for an interaction is taken by the source of the arrow.

DISCOVERY Interaction: a Discovering Node discovers Sleepy Node(s) via Proxy or Discovery Server; for example:

- A Discovering Node wants to discover given services related to a group of deployed sensors by sending a multicast to /.well-known/core. It gets responses for the sleeping sensors from the Proxy nodes.
- During commissioning phase, a discovering node queries a Discovery Server to find all the proxies providing a given service.

REPORT Interaction: On request of a Destination Node or because of configuration settings which have instructed the Node to do so, a Node sends a sequence of event notifications to destination Node(s), (A) directly or (B) via Proxy; for example:

- A battery-powered sensor sends a notification with "battery low" event directly to a designated Destination Node (REPORT(A)).
- A battery-powered occupancy sensor detects an event "people present", switches on the radio and multicasts an "ON" command to a group of lights (REPORT(A)).
- A battery-powered temperature sensor reports periodically the room temperature to a proxy Node (REPORT(A)). The proxy node reports to all associated HVAC destination nodes when the temperature change deviates from a predefined range (REPORT(B)).

WRITE Interaction: A node sends a request to a proxy to set a value.

- o A Sleepy Node WRITES to the proxy; for example:
 - A battery-powered sensor wants to extend the registration lifetime of its delegated resource at the Proxy.
- o A configuring Node WRITES information to a Proxy; for example:
 - A configuring Node changes the reporting frequency of a deployed sensor by contacting the Proxy node to which the sensor is registered.
 - Sensor firmware is upgraded. A configuring Node pushes firmware data blocks to the Proxy, which pushes the blocks to the Sleepy Node.
 - A configuring Node adds a new subscription to an operational sensor via the Proxy. From that moment on, the new Node receives also the sensor events and status updates from the sensor.

READ Interaction: A node sends a read request to a node that returns a value.

- o Sleepy Node sends a read request to a server Node; for example:
 - A sensor (periodically) updates internal data tables by fetching it from a predetermined remote node.
 - A sensor (periodically) checks for new firmware with a remote node. If new firmware is found, the sensor switches to a non-sleepy operation mode, and fetches the data.
- o A Sleepy Node sends a read request to its proxy; for example:

- A sensor (periodically) checks with his Proxy availability of configuration updates or changes of its delegated resources (e.g. a sensor may detect in this way that a configuring Node has changed its name or modified its reporting frequency).
- o A reading Node sends a read request to a proxy; for example:
 - A Node (e.g. in the backend) requests the status of a deployed sensor, e.g. asking the sensor state and/or firmware version and/or battery status and/or its error log. The Proxy returns this information.
 - A Node requests a Proxy when a Sleepy sensor was 'last active' (i.e. identified as being awake) in the network.

2.2. Architecture

The architecture associated with the support of Sleepy Nodes is illustrated in Figure 2. Three High level interfaces are shown.

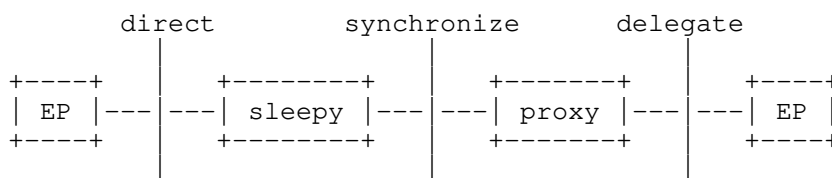


Figure 2: Architecture of Sleepy Node support

- o Direct interface: it allows the Sleepy Node to communicate directly to endpoints (i.e. for sending or reading information). The operations performed via this interface are always initiated by the Sleepy Node when its sleep period ends.
- o Delegate interface: via this interface the Proxy exposes the values of delegated resources to interested endpoints on behalf of the Sleepy Node. The same interface is used by endpoints which want to communicate with the Sleepy Node (e.g. for reading or writing information).
- o Synchronize interface: used by Sleepy Node and Proxy to synchronize values of delegated resources. Through this interface operations as discovery of the Proxy, registration, initialization and update of resources at the Proxy are performed, along with a de-registration operation to explicitly remove resources already registered to the Proxy.

The interfaces consist of a set of functions which together realize the interactions described in Section 2.1.

Endpoints and the proxy communicate with a Resource Directory (RD) to discover resources of the Sleepy Node and delegated resources on the proxy (not shown in the Figure 2).

2.3. Example contents

The examples presented in this specification make use of a smart temperature sensor the resources of which are defined below using Link Format [RFC6690]. Three resources are dedicated to the Device Description (manufacturer, model, name) and one contains the current temperature in degree Celsius.

```
</dev/mfg >;rt="ipso.dev.mfg";if="core.rp",  
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",  
</dev/n>;rt="ipso.dev.n";if="core.p",  
</sen/temp>;rt="ucum.Cel";if="core.s"
```

3. Design motivation

The Sleepy Node stack features a CoAP interface, to make the Sleepy Node part of the IP-based network. Adding CoAP with a transport protocol increases the possibilities to configure the Sleepy Node within the network. The increased energy consumption coming from the overhead of the CoAP and IP headers can be acceptable in many cases.

The proxy and Sleepy Node make use of the /.well-known/core resource to handle discovery during network initialization. Using the Resource Directory during operation of the Sleepy Node reduces its participation in the discovery traffic.

A Sleepy Node delegates its resources to a proxy. The proxy functionality extends the functionality of the RD, because the proxy handles the value of the resource, and the RD does not. A proxy may support multiple Sleepy Nodes. A Sleepy Node may also delegate its resources to multiple proxies. A node can select a proxy that handles the resource of the Sleepy Node of choice.

The complexity of the discovery and delegation interfaces is minimized by reusing the RD interface as much as possible.

4. Interactions involving Resource Directory

It is assumed that the Proxy has a resource type rt="core.sp", where sp stands for sleepy proxy.

In order to become fully operational in a network and to communicate over the functional interfaces shown in Figure 2, a Sleepy Node and the Proxy need to perform operations via the Registration interface of the RD:

- Discovery of Proxy via RD. The Sleepy Node MAY discover the Proxy by sending a request to the RD to return all EP with `rt=core.sp`.
- Register existence of Proxy. When a RD is present and a Sleepy Node has registered itself to a Proxy (see Section 5.2), the Proxy MUST register the Sleepy Node at the RD and MUST keep this registration up-to-date.
- Register delegated resources. When a RD is present, the Proxy MUST register the delegated resources at the RD and keep them up-to date.

A Configuring Endpoint (often part of a so-called Commissioning Tool) registers the services that are reported directly by the Sleepy Node in the resource directory, by registering the resource type and the multicast address. The multicast address can be associated with a group as described in [I-D.ietf-core-resource-directory].

A discovering Endpoint can discover one or more Sleepy Node resources via the Resource Directory.

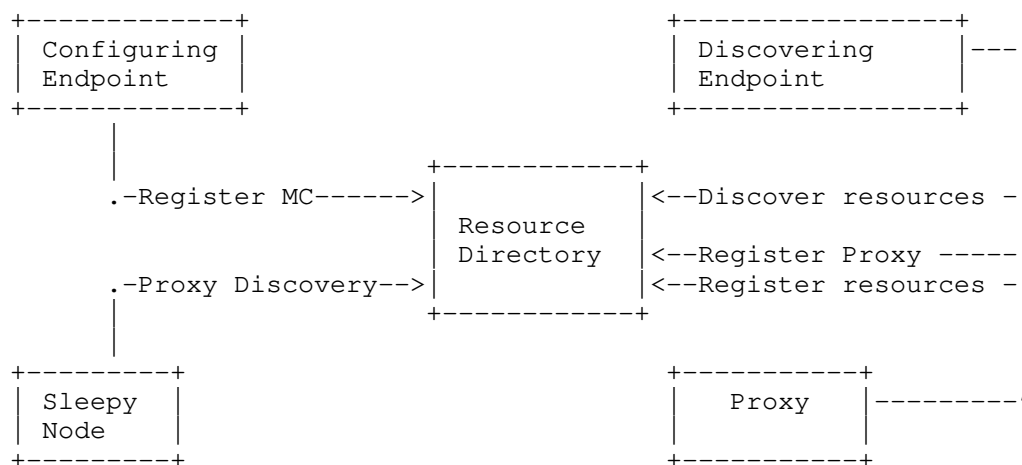


Figure 3: Interactions involving Resource Directory

5. Synchronize interface

The functions of the synchronize interface implemented by the Proxy are described in this section.

5.1. Sleepy Node discovers proxy

A Sleepy Node can discover the proxy in two ways:

- via the CoAP interface [RFC7390] by sending a multicast message to discover an endpoint with `rt=core.sp`.
- via RD as already described in Section 4.

The following example shows a sleeping endpoint discovering a proxy using this interface, thus learning that the base Proxy resource, where the Sleepy Node resources are registered, is at `/sp`.

Sleepy	<pre> ----- GET /.well-known/core?rt=core.sp -----> <----- 2.05 Content "</sp>; rt="core.sp" ----- </pre>	Proxy
--------	--	-------

```

Req: GET coap://[ff02::1]/.well-known/core?rt=core.sp
Res: 2.05 Content
</sp>;rt="core.sp"

```

The use of `/sp` is recommended and not compulsory.

5.2. Registration at a Proxy

Once a Sleepy Node has discovered a Proxy by means of one of the procedures described in Section 5.1, the registration step can be performed. To perform registration, a Sleepy Node sends to the Proxy Node a CoAP POST request containing a description of the resources to be delegated to the Proxy as the message payload in the CoRE Link Format [RFC6690]. The description of the resource includes the Sleepy Node identifier, its domain and the lifetime of the registration.

Upon successful registration a Proxy creates a new delegated resource or updates an existing delegated resource and returns its location. The resources specified by the Sleepy Node during registration are created with path that has as prefix the base Proxy resource path (e.g. `/sp`). The registration interface **MUST** be implemented to be

idempotent, so that registering twice with the same endpoint parameter does not create multiple delegated resources. The delegated resource SHOULD implement the Interface Type CoRE Link List defined in [I-D.ietf-core-interfaces]. A GET request on this resource MUST return the list of delegated resources for the corresponding Sleepy Node.

After successful registration, a Proxy SHOULD enable resource discovery for the new resources by updating its `"/.well-known/core"` resource. A Proxy MUST wait for the initial representation of a resource before it can be visible during resource discovery. The top level delegated resource MUST be published in `"/.well-known/core"` to enable the discovery of the resources via RD as described in Section 4. Resources of a delegated container SHOULD be discoverable either directly in `"/.well-known/core"` or indirectly through gradual reveal from the delegated resource. The Web Link of a delegated resource MUST contain an `"ep"` attribute with the value of the End-Point parameter received during registration.

A Proxy MAY be configured to register the Sleepy Node's resources in a RD. In this case, a Sleepy Node MUST NOT register the resources in a RD by itself since it is the responsibility of the Proxy to perform the registration in the RD on behalf of the Sleepy Node. Since each Sleepy Node may register resources with different lifetimes, a Proxy MUST register the resources of a given Sleepy Node in a dedicated path of the RD.

In case a Sleepy Node delegates its own resources to more than one Proxy and each Proxy registers the Sleepy Node's resource in a RD, the RD entries from the different Proxies for the same Sleepy Node risk to overlap.

To avoid this problem, a Proxy MUST create its own resource path to register the resources of a Sleepy Node on the RD.

The new path name is typically formed by concatenating the Proxy's endpoint identifier with the path in use. This precaution ensures that the `ep` identifier of a Sleepy Node is unique for each resource path in the RD.

Implementation note: It is not recommended to reuse the value of the `ep` parameter in the URI of the delegated resource. This parameter may be a relatively long identifier to guarantee global uniqueness (e.g. EUI64) and would generate inefficient URIs on the Proxy where only a local handler is necessary.

The following example shows a Sleepy Node registering with a Proxy.

Sleepy	Proxy
<pre> --- POST /sp?ep=0224e8fffe925dcf;rt=sensor "</dev..."---> <-- 2.01 Created Location: /sp/0 ----- </pre>	<pre> </pre>

```

Req: POST coap://sp.example.org/sp?ep=0224e8fffe925dcf;rt=sensor
Etag: 0x3f
Payload:
</dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</dev/n>;rt="ipso.dev.n";if="core.p",
</sen/temp>;rt="ucum.Cel";if="core.s"

Res: 2.01 Created
Location: /sp/0

```

The delegated resource has been created with path /sp/0 on the Proxy in the example above. The path to the ep can be discovered as shown below:

```

Req: GET coap://sp.example.org/.well-known/core
Res: 2.05 Content
</sp>;rt="core.sp",
</sp/0>;ep="0224e8fffe925dcf";rt="sensor"

```

A node can discover the delegated resources of the ep as shown below:

```

Req: GET coap://sp.example.org/sp/0
Res: 2.05 Content
Payload:
</sp/0/dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</sp/0/dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</sp/0/dev/n>;rt="ipso.dev.n";if="core.p",
</sp/0/sen/temp>;rt="ucum.Cel";if="core.s"

```

Once the resources are registered in the Proxy, the Proxy registers the delegated resources in the RD.

Proxy	RD
<pre> --- POST /rd?ep=0224e8fffe925dcf "</sp/0..." -----> <-- 2.01 Created Location: /rd/6534 ----- </pre>	<pre> </pre>

Req: POST coap://rd.example.org/rd?ep=0224e8fffe925dcf

Etag: 0x6a

Payload:

```

</sp/0/dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</sp/0/dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</sp/0/dev/n>;rt="ipso.dev.n";if="core.p",
</sp/0/sen/temp>;rt="ucum.Cel";if="core.s"

```

Res: 2.01 Created

Location: /rd/6534

5.3. De-registration at a Proxy

Sleepy Node resources in the Proxy are kept active for the period indicated by the lifetime parameter. The Sleepy Node is responsible for refreshing the delegated resource within this period using either the registration or update function (see Section 5.5 of the Synchronize interface). Once a delegated resource has expired, the Proxy deletes all resources associated to that resource and updates its `"/.well-known/core"` resource. When the Proxy resources are also registered in a RD, the RD and delegated resources are supposed to have the same lifetime. Consequently, when the delegated resource expires, a Proxy MAY let the RD resource expire too instead of explicitly deleting it. When the delegated resource is deleted by means of explicit de-registration operation then also the RD resource MUST be explicitly removed.

A Proxy could lose or delete the delegated resource associated to a Sleepy Node without sending an explicit notification (e.g. after reboot). A Sleepy Node SHOULD be able to detect this situation by processing the response code while using the Sleepy Node Operation or Update interface. Especially an error code `"4.04 Not Found"` SHOULD cause the Sleepy Node to register again. A Sleepy Node MAY also register with multiple proxies to alleviate the risk of interruption of service.

5.4. Initialization of delegated resource

Once registration has been successfully performed, the Sleepy Node must initialize the delegated resource. To send the initial contents (e.g. values, device name, manufacturer name) of the delegated resources to the Proxy, the Sleepy Node uses CoAP PUT repeatedly.

The basic interface is specified as follows:

Interaction: Sleepy -> Proxy

Method: PUT

URI Template: /{+location}{+resource}{?lt}

URI Template Variables:

location := This is the Location path returned by the Proxy as a result of a successful registration.

resource := This is the relative path to a delegated resource managed by the registered Sleepy Node.

lt := Lifetime (optional). The number of seconds by which the lifetime of the whole delegated resource is extended. Range of 1-4294967295. If no lifetime is included, the current remaining lifetime stays unchanged.

Request Content-Type: Defined at registration

Response Content-Type: Defined at registration for GET method.
application/link-format for PUT method if at least one of the mutable resources has been updated since the last PUT request.

Etag: The Etag option MAY be included to allow clients to validate a resource on multiple Proxies.

Success: 2.01 "Created", the request MUST include the initial representation of the delegated resource.

Success: 2.04 "Changed", the request MUST include the new representation of the delegated resource.

Success: 2.05 "Content", the response MUST include the current representation of the delegated resource.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example describes how a Sleepy Node can initialize the resource containing its manufacturer name just after registration.

Sleepy	Proxy
--- PUT /sp/0/dev/mfg "acme" ----->	
<-- 2.01 Created -----	

Req: PUT /sp/0/dev/mfg
 Payload: acme
 Res: 2.01 Created

The example below shows how a Sleepy Node can indicate that it is supposed to send a temperature value at least every hour to keep its delegated resource active.

Sleepy	Proxy
--- PUT /sp/0/sen/temp?lt=3600 "22" ----->	
<-- 2.04 Changed -----	

Req: PUT /sp/0/sen/temp?lt=3600
 Payload: 22
 Res: 2.04 Changed

The use of repeated CoAP PUT can be avoided by writing all relevant resources into the Proxy in one operation by means of the Batch interface described in [I-D.ietf-core-interfaces]. After successful initialization, a Proxy SHOULD enable resource discovery for the new delegated resources by updating its /.well-known/core resource.

5.5. Sleepy Node updates delegated resource at Proxy

A Sleepy Node can update a delegated resource at the Proxy (REPORT A) using standard CoAP PUT requests on the delegated resource as shown in Section 5.4.

When a Sleepy Node sends a PUT request to update its resources, the response MAY contain a link-format payload. The payload does not

directly relate to the target resource of the PUT request. Instead, it is a list of web links to resources that have been modified by clients since either the last PUT request or the last call to the modification check interface (see Section 5.6).

5.6. Sleepy Node READs resource updates from Proxy

This function allows a Sleepy Node to retrieve a list of delegated resources that have been modified at the Proxy by other nodes. The interface format for GET is the same as the one specified for PUT in Section 5.4.

A configuring Node (EP) can update a resource in the Proxy. The Sleepy Node receives an indication of the changed resources as specified in Section 5.5.

The Sleepy Node can send GET requests to its Proxy on each delegated resource in order to receive their updated representation. The example in Figure 4 shows a configuration node which changes the name of a Sleepy Node at the Proxy. The Sleepy Node can then check and read the modification in its resource.

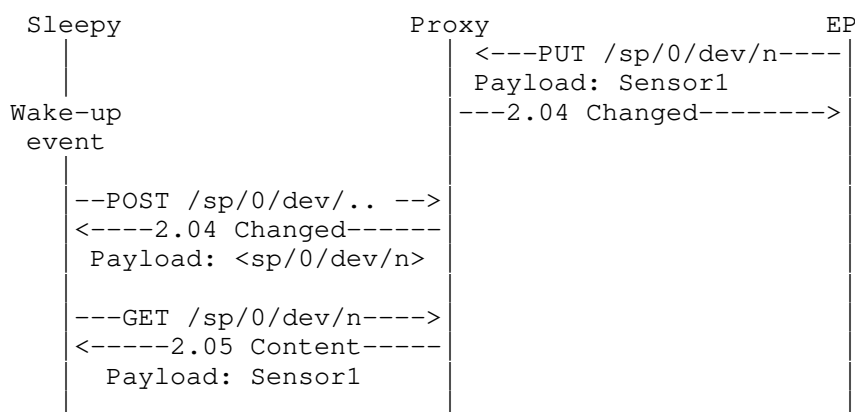


Figure 4: Example: A Sleepy Node READs resource updates from his Proxy

6. Delegate Interface

This section details the functions belonging to the delegate interface.

6.1. Discovering Endpoint discovers Sleepy Node at Proxy

Through this function, a Discovering Endpoint can discover one or more Sleepy Node(s) at a Proxy. In case a Resource Directory is not present, this is the only way to discover Sleepy Nodes. A CoAP client discovers resources owned by the Sleepy Node but hosted on the Proxy using typical mechanisms such as one or more GETs on the resource `/.well-known/core` [RFC6690].

Resource discovery between an Endpoint and a proxy or an Endpoint and a RD needs special care to take into account the fact that resources from a Sleepy Node might appear duplicated. EPs SHOULD employ 2-step resource discovery by looking up Sleepy Nodes AND resource types to detect duplicate resources. EPs MAY use single-step resource discovery only if the Sleepy Node can register with no more than one Proxy. An EP can use the "ep" link attribute as a filter on the `/.well-known/core` resource to retrieve a list of endpoints and detect duplicate Sleepy Nodes registered on multiple proxies. An EP can use the "ep" type of lookup to do the same on a RD. The result of endpoint discovery is then used to filter out duplicate resources returned from simple resource discovery.

The following example shows a client discovering the Sleepy Nodes and learning that the Sleepy Node 0224e8fffe925dcf is registered on two Proxies.

EP	proxy1	proxy2
----- GET /.well-known/core?ep=* ----->	----->	----->
<----- 2.05 Content "</sp/0>..." -----	-----	-----
<----- 2.05 Content "</sp/0>..." -----	-----	-----

Req: GET coap://[ff02::1]/.well-known/core?ep=*

Res: 2.05 Content

</sp/0>;ep="0224e8fffe925dcf"

Res: 2.05 Content

</sp/0>;ep="02004cffffe4f4f50"

</sp/1>;ep="0224e8fffe925dcf"

From the previous exchange and the next resource discovery request, the EP can infer that the resources `coap://sp1/sp/0/sen/temp` and `coap://sp2/sp/1/sen/temp` actually come from the same Sleepy Node with `ep=0224e8fffe925dcf`.

EP	proxy1	proxy2
- GET /.well-known/core?rt=ipso:ucum.Cel ->	----->	
<---- 2.05 Content "</sp/0>..." -----		
<---- 2.05 Content "</sp/1>..." -----		


```

Req: GET coap://[ff02::1]/.well-known/core?rt=ucum.Cel
                                     &ep=0224e8fffe925dcf
Res: 2.05 Content
</sp/0/sen/temp;rt="ucum.Cel"
Res: 2.05 Content
</sp/1/sen/temp>;rt="ucum.Cel"

```

6.2. Proxy REPORTs events to Endpoint

This interface can be used by the Endpoint to receive event report message to Proxy (REPORT A) which further notifies it to interested Destination Endpoint(s) (REPORT B). This indirect reporting is useful for a scalable solution, e.g. there may be many interested subscribers but the Sleepy Node itself can only support a limited number of subscribers given its limits on battery energy. A client interested in the events related with a specific resource may send a CoAP GET to the Proxy, to obtain the last published state. If a Reading node is interested in receiving updates whenever the Sleepy Node reports new event to its Proxy, it can use observe [I-D.ietf-core-observe] at the Proxy for that specific resource.

A proxy using the CoAP protocol [RFC7252] SHOULD accept to establish a CoAP observation relationship between the delegated resource and a client as defined in [I-D.ietf-core-observe].

A Sleepy Node may stop updating its delegated resources without explicitly removing its delegated resource (e.g. transition to another proxy after network unreachability detection). An Endpoint can detect this situation when the corresponding delegated resource has expired. Upon receipt of a response with error code 4.04 "Not Found", an Endpoint SHOULD restart resource discovery to determine if the resources are now delegated to another proxy.

The interface function is specified as follows:

Interaction: EP -> Proxy

Method: Defined at registration

URI Template: `/{"location"}{"resource"}`

URI Template Variables:

`location` := This is the Location path returned by the Proxy as a result of a successful registration.

`resource` := This is the relative path to a delegated resource managed by a Sleepy Node.

Content-Type: Defined at registration

In the example below an EP observes the changes of temperature through the Proxy.

Sleepy	Proxy	EP
	<code><- GET /sp/0/sen/temp -</code>	
	<code>(observe)</code>	
	<code>-- 2.05 Content "22" -></code>	
<code>- PUT /sp/0/sen/temp "23" -></code>		
<code><- 2.04 Changed -----</code>		
	<code>-- 2.05 Content "23" -></code>	

6.3. A Node WRITES to Sleepy Node via Proxy

A Configuring Node uses CoAP PUT to write information (such as configuration data) to the Proxy, where the information is destined for a Sleepy Node. Upon change of a delegated resource, an internal flag is set in the Proxy that the specific resource has changed. Next time the Sleepy Node wakes up, the Sleepy Node checks the Proxy for any modification of its delegated resources and reads those changed resources using CoAP GET requests, as shown in Figure 4. The allowed resources that a Configuring Node can write to, and the CoAP Content-Format of those CoAP resources, is determined in the initial registration phase.

The following example shows a commissioning tool (EP) changing the name of a Sleepy Node through a Proxy. The Sleepy Node detects this change right after updating its current temperature.

Sleepy	Proxy	EP
	<-- PUT /sp/0/dev/n ---	
	-- 2.04 Changed ----->	
- PUT /sp/0/sen/temp --->		
<- 2.04 Changed -----		
Payload: <sp/0/dev/n> ---		
- GET /sp/0/dev/n ----->		
<- 2.05 Content -----		

Req: PUT /sp/0/dev/n

Payload: "sensor-1"

Res: 2.04 Changed

Req: PUT /sp/0/sen/temp

Payload: "24"

Res: 2.04 Changed, Content-Type: application/link-format

Payload: "</sp/0/dev/n>"

Req: GET /sp/0/dev/n

Res: 2.05 Content

Payload: "sensor-1"

6.4. A Node READs information from Sleepy Node via Proxy

A Reading Node uses standard CoAP GET to read information of a Sleepy Node via a Proxy. However, not all information/resources from the Sleepy Node may be copied to the Proxy. In that case, the Reading Node cannot get direct access to resources that are not delegated to the Proxy. The strategy to follow in that case is to first WRITE to the Sleepy Node (via the Proxy, Section 6.3) a request for reporting this missing information; where the request can be fulfilled by the Sleepy Node the next time the Sleepy Node wakes up.

7. Direct Interface

This section details the functions belonging to the direct interface.

7.1. Sleepy Node REPORTs events directly to Destination Node

When the Sleepy Node needs to report an event to Destination nodes or groups of Destination nodes present in the subscribers list, it

becomes Awake and then it can use standard CoAP POST unicast or multicast requests to report the event.

TODO: MC example

7.2. A Sleepy Node READs information from a Server Node

A Sleepy Node while Awake uses standard CoAP GET to read any information from a Server Node. While the Sleepy Node awaits a CoAP response containing the requested information, it remains awake. To increase battery life of Sleepy Nodes, such an operation should not be performed frequently.

8. Realization with PubSub broker

The PubSub broker [I-D.koster-core-coap-pubsub] can be used to implement the REPORT function of the Sleepy Node proxy specified in this document. However, there are some differences to be taken into account:

- The PubSub broker handles topics. In the case of the proxy the topics must be equated to resources.
- Clients publish anonymously updates to a topic. In the case of the proxy, a delegated resource is bound to one given node that is allowed to update it. For the same functionality, the PubSub broker must restrict topic updates to one client only. The client linked to the topic must be visible to the clients which subscribe to the topic.

In addition, some other functionality needs to be added to the PubSub broker to satisfy the interaction model shown in Figure 1:

- the READ function from Sleepy Node to proxy is not covered by the PubSub broker. The PubSub broker needs to piggy-back a "check topic" on the confirmation of a publication by the proxy. The proxy can then perform a Read on the signalled topic.
- The interaction "register resources" from proxy to Resource Directory, shown in Figure 3, is not part of the PubSub broker.

9. IANA Considerations

The new Resource Type (rt=) Link Target Attribute, 'core.sp' needs to be registered in the "Resource Type (rt=) Link Target Attribute Values" sub registry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

10. Security Considerations

For the communication between Sleepy Node and Proxy it MAY be sufficient to use Layer 2 (MAC) security without the recommended use of DTLS. However, it must be ascertained that the Sleepy Node can communicate only with a given secured Proxy. A Sleepy Node may obtain the Layer 2 network key using the bootstrapping mechanism described in [I-D.kumar-6lo-selective-bootstrap]. DTLS MUST be used over link-layer security for further transport-layer protection of messages between Regular Nodes and Proxies in the network. There are no special adaptations needed of the DTLS handshake to support Sleepy Nodes. During the whole handshake, Sleepy Nodes are required to remain awake to avoid that, in case of small retransmission timers, the other node may think the handshake message was lost and starts retransmitting. In view of this, the only key point, therefore, is that DTLS handshakes are not performed frequently to save on battery power. Based on the DTLS authentication, also an authorization method could be implemented so that only authorized nodes can e.g.

- Act as a Proxy for a Sleepy Node. (The Proxy shall be a trusted device given its important role of storing values of parameters for the delegated resources);
- READ data from Sleepy Nodes;
- WRITE data to Sleepy Nodes (via the Proxy);
- Receive REPORTs from Sleepy Nodes (direct or via Proxy).

11. Acknowledgements

Much of the text and examples in this document are copied from [I-D.vial-core-mirror-server]. Matthieu Vial has generously authorized us to use his text. Rahman Akbar has pointed out the CoAP dependency of earlier versions.

12. Changelog

RFC editor, please delete this section before publication.

From version 2 to version 3:

Introduced interfaces and copied examples and text from mirror server draft.

From version 3 to version 4:

Comparison with PubSub Broker completed.

Mistakes in examples removed.

Less dependence on 6LowPAN networks.

Added Design motivation section.

13. References

13.1. Normative References

- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-16 (work in progress), December 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<http://www.rfc-editor.org/info/rfc7390>>.

13.2. Informative References

- [I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., and M. Koster, "CoRE Interfaces", draft-ietf-core-interfaces-03 (work in progress), July 2015.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-04 (work in progress), July 2015.

[I-D.koster-core-coap-pubsub]

Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-02 (work in progress), July 2015.

[I-D.kumar-6lo-selective-bootstrap]

Kumar, S. and P. Stok, "Security Bootstrapping over IEEE 802.15.4 in selective order", draft-kumar-6lo-selective-bootstrap-00 (work in progress), March 2015.

[I-D.vial-core-mirror-server]

Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-server-01 (work in progress), April 2013.

[RFC6763]

Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<http://www.rfc-editor.org/info/rfc6763>>.

Authors' Addresses

Teresa Zotti
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
The Netherlands

Phone: +31 6 21175346
Email: teresa.zotti@philips.com

Peter van der Stok
Consultant

Phone: +31 492474673
Email: consultancy@vanderstok.org

Esko Dijk
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
The Netherlands

Phone: +31 6 55408986
Email: esko.dijk@philips.com