ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

O. Bergmann
S. Gerdes
Universitaet Bremen TZI

October 19, 2015

Using The Delegated CoAP Authentication and Authorization Framework (DCAF) With CBOR Encoded Message Syntax
draft-bergmann-ace-dcaf-cose-00

Abstract

This specification defines a profile for the Delegated CoAP Authentication and Authorization Framework (DCAF) that facilitates client authentication and authorization in a constrained environment using the CBOR Encoded Message Syntax.

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.

Table of Contents

1. Introduction 2
1.1. Terminology 3
2. Overview 4
2.1. Sending Authorized Requests 4
2.2. Responding to an Authorized Request 5
3. Establishing a Security Context 6
3.1. Unauthorized Resource Request Message 6
3.2. SAM Information Message 7
3.3. Access Request 8
3.4. Ticket Request Message 10
3.5. Ticket Grant Message 10
3.6. Ticket Transfer Message 11
3.7. Security Association between C and S 12
4. Piggybacked Protected Content 13
5. CoAP Options Authorization and Authorization-Format 14
6. Canonicalization of the CoAP Message Header 14
7. The "auth-info" Link Relation 16
8. Security Considerations 17
9. IANA Considerations 17
9.1. CoAP Option Registration 17
10. Acknowledgements 17
11. References 17
11.1. Normative References 17
11.2. Informative References 18
Authors' Addresses 19

1. Introduction

The Delegated CoAP Authentication and Authorization Framework (DCAF) is designed to be agnostic of the actual mechanism being used to secure the communication between the ACE actors. While the original specification [I-D.gerdes-ace-dcaf-authorize] defines how to use DCAF messaging for establishing a Datagram Transport Layer Security (DTLS) [RFC6347] channel between actors on the constrained level (cf. [I-D.ietf-ace-actors]), this document specifies a binding of DCAF to the CBOR Encoded Message Syntax, COSE [I-D.ietf-cose-msg].

To reduce confusion, we use "DTLS DCAF" to refer to DCAF based on DTLS security, and "COSE DCAF" to refer to DCAF as defined in the present document.

DCAF defines authorized access to a resource hosted on a resource Server (S) based on a security context that is established between the requesting Client (C) and S. In DTLS DCAF, this security context is tied to a DTLS channel which allows for end-to-end integrity protection and confidentiality of communication. In the presence of

intermediaries such as, e.g., CoAP proxies, channel security may not be applicable for all configurations. If this is the case, the exchanged information must be protected at the application level to help achieving the principals' security requirements. The IETF working group CBOR Object Signing and Encryption (COSE) has defined a Concise Binary Object Representation (CBOR) [RFC7049] representation for signed and encrypted objects as well as message authentication codes.

This specification uses this CBOR Encoded Message Syntax [I-D.ietf-cose-msg] to protect the DCAF protocol flow on the application level. The features of this DCAF profile are:

- Authenticated exchange of authorization information.
- Simplified authentication on constrained nodes by handing the more sophisticated authentication work over to less-constrained devices.
- Support of secure constrained device to constrained device communication.
- Authorization policies of the principals of both participating parties are ensured.
- Simplified authorization mechanism for cases where implicit authorization is sufficient.
- Can be made to work just using symmetric encryption on the constrained nodes.
- Enable delivery of piggybacked protected content as discussed in [I-D.gerdes-ace-dcaf-authorize].

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 RFC 2119 [RFC2119].

Readers are expected to be familiar with the terms and concepts defined in [I-D.gerdes-ace-dcaf-authorize].

2. Overview

This specification retains the most important features of DCAF by utilizing the same basic messaging mechanism. DCAF ensures that protected information is accessible only by authorized entities, i.e. access must be authenticated and the principal that oversees the particular piece of information must permit the requested action. The DCAF specification cryptographically ties this authorization to a DTLS session setup between the communicating Client and Server. The DTLS key material used for creating this session hence defines the security context between the communicating parties. By supplementing DCAF with the notion of a context identifier, the same mechanism can be used with application level security as well.

2.1. Sending Authorized Requests

In general, every request that C sends to S must be treated by S within a particular security context with C. [_1] If S is not able to otherwise identify the security context from the message context, the context identifier must be transferred within the respective message. An example of a request containing an explicit context identifier is shown in Figure 1 using the CBOR diagnostic notation as defined in Section 6 of [RFC7049] to describe the actual data represented in CBOR.

```
PUT /r
Content-Format: application/cose+cbor
[ h'a10300',                    # protected { content_type: text/plain }
  { alg: HMAC 256/256,          # unprotected
    kid: h'3233386473613239'    # context identifier: "238dsa29"
  },
  h'48656c6c6f20576f726c6421',  # payload: "Hello World!\n"
  h'....',                # tag: HMAC(options+protected+payload, secret)
  [ [ h'', {}, h'' ] ]          # recipients
]
```

Figure 1: Example for CoAP Request with Explicit Context Identifier

Figure 1 shows a PUT request from C to S for resource r containing a payload of type 'application/cose+cbor' that carries a COSE_Mac structure to integrity-protect the request using the MAC key from a previously established security context with identifier '238dsa29'. As the security context can be determined from the context identifier, an empty COSE_recipient structure is used. Note that the integrity protection not only covers the message payload but also the content type and various sensitive CoAP options such as Uri-Path that will be passed to the MAC creation functions as canonicalized external_aad as described in Section 6.

Note1: Where confidentiality is required, a COSE_encryptData structure will be used instead of the COSE_Mac structure.

Note2: COSE_enveloped may be used instead of COSE_encryptData when dynamically generated session keys should be used, e.g. with protected piggybacked content.

Note3: As transferring COSE objects as the CoAP message payload is not always possible (e.g. in GET requests), this specification defines two new CoAP options 'Authorization' and 'Authorization-Format' that can be used to convey the authorization information.

To retrieve a resource representation using the request method GET, the authorization information is conveyed in an Authorization attribute as shown in Figure 2.

```
GET /r
Authorization: [ h'',           # protected (empty)
  { alg: HMAC 256/256,          # unprotected
    kid: h'3233386473613239'    # context identifier: "238dsa29"
  },
  nil,                          # payload (empty)
  h'....',                      # tag: HMAC(options+protected, secret)
  [ [ h'', {}, h'' ] ]          # recipients
]
```

Figure 2: Example for CoAP Request with Authorization Option

The request in Figure 2 uses the default Authorization-Format 'application-cose' for the contents of the Authorization option which is a COSE_Mac structure. As in Figure 1 the MAC key from a previously established security context with identifier '238dsa29' is used and an empty COSE_recipient structure is used. The integrity protection for this request not only covers the message payload but also the content type and various sensitive CoAP options such as Uri-Path that will be passed to the MAC creation functions as external_aad. The external_aad MUST be constructed as CBOR bytes containing a canonicalized CoAP message as specified in Section 6.

2.2. Responding to an Authorized Request

A response to an Authorized Request that uses this DCAF profile MUST be protected according to the principals' security objectives covered by the existing security context between C and S. Usually, this means that a resource representation returned by S in the response is wrapped into a COSE_encryptData or COSE_enveloped structure. A protected response to an authorized GET request is depicted in Figure 3.

Note: For AEAD ciphers, confidentiality and integrity can be achieved in one encryption step. For other cipher suites, it may be more convenient to use a COSE_Mac structure when only message integrity is required.

```
2.05 Content
Content-Format: application/cose+cbor
[ h'a10300',               # protected { content_type: text/plain }
  { alg: AES-CCM-16-64-128,        # unprotected
    nonce: h'77cd8a8047b7af7113bb074bcc', # nonce
  },
  h'TBD:encrypted payload w/ tag', # ciphertext
  # recipients:
  [ [ h'',                         # protected (absent for AE alg.)
      { alg: A128KW,               # unprotected
        kid: h'3233386473613239'   # context identifier: "238dsa29"
      },
      h'fec31142bc...'             # encrypted session key
  ] ]
]
```

Figure 3: Example for a Protected Response Containing a Resource Representation

3. Establishing a Security Context

Section 2.1 illustrates the use of CBOR Encoded Message Syntax for sending Authorized Requests and Responses. Before this communication can take place the security context must be established using the COSE DCAF message types as described in this section. This section describes the basic message flow as outlined in [I-D.gerdes-ace-dcaf-authorize], but using the CBOR Encoded Message Syntax to convey the DCAF information instead of DTLS.

3.1. Unauthorized Resource Request Message

The optional Unauthorized Resource Request message is a request for a resource hosted by S for which no proper authorization has been granted so far. S MUST treat any CoAP request as an Unauthorized Resource Request message when any of the following holds:

- The request has been received unprotected.
- The security context for the received request is unknown.
- S has no valid access ticket for the sender of the request regarding the requested action on that resource.

- o S has a valid access ticket for the sender of the request, but this does not allow the requested action on the requested resource.

Note: These conditions ensure that S can handle requests autonomously once access has been granted and a security context has been established between C and S.

Unauthorized Resource Request messages MUST be denied with a client error response. In this response, the Server MUST provide proper SAM Information to enable the Client to request an access ticket from S's SAM as described in Section 3.2. S MAY include a protected piggybacked response with the SAM Information Message in the Unauthorized Resource Request message, as discussed in Section 4.

The response code MUST be 4.01 (Unauthorized) in case the sender of the Unauthorized Resource Request message is not authenticated, or if S has no valid access ticket for C. If S has an access ticket for C but not for the resource that C has requested, S MUST reject the request with a 4.03 (Forbidden). If S has an access ticket for C but it does not cover the action C requested on the resource, S MUST reject the request with a 4.05 (Method Not Allowed).

Note: The use of the response codes 4.03 and 4.05 is intended to prevent infinite loops where a naive Client optimistically tries to access a requested resource with any access token received from the SAM. As malicious clients could pretend to be C to determine C's privileges, these detailed response codes must be used only when a certain level of security is already available which can be achieved only when the Client is authenticated.

3.2. SAM Information Message

The SAM Information Message is sent by S as a response to an Unauthorized Resource Request message (see Section 3.1) to point the sender of the Unauthorized Resource Request message to S's SAM. The SAM information is a set of attributes containing a URI that specifies the SAM in charge of S.

An optional field A lists the different content formats that are supported by S.

The message MAY also contain a timestamp generated by S.

Figure 4 shows an example for a SAM Information message payload using stylized CBOR diagnostic notation. (Refer to [I-D.gerdes-ace-dcaf-authorize] for a detailed description of the available attributes and their semantics.)

```
4.01 Unauthorized
Content-Format: application/dcaf+cbor
{SAM: "coaps://sam.example.com/authorize", TS: 168537,
A: [ ct_cose_msg ] }
```

Figure 4: SAM Information Payload Example

In this example, the attribute SAM points the receiver of this message to the URI "coaps://sam.example.com/authorize" to request access permissions. The originator of the SAM Information payload (i.e. S) uses a local clock that is loosely synchronized with a time scale common between S and SAM (e.g., wall clock time). Therefore, it has included a time stamp on its own time scale that is used as a nonce for replay attack prevention.

The content format accepted by S is 'application/cose+cbor' defined in [I-D.ietf-cose-msg] to indicate DCAF over CBOR Encoded Message Syntax as defined in this document.

Editorial note: ct_cose_msg is to be replaced with the numeric value assigned for 'application/cose+cbor'.

The examples in this document are written in CBOR diagnostic notation to improve readability. Figure 5 illustrates the binary encoding of the message payload shown in Figure 4.

```
a2                                  # map(2)
    00                              # unsigned(0) (=SAM)
    78 21                           # text(33)
       636f6170733a2f2f73616d2e6578
       616d706c652e636f6d2f617574686f72
       697a65             # "coaps://sam.example.com/authorize"
    05                              # unsigned(5) (=TS)
    1a 00029259                     # unsigned(168537)
    0a                              # unsigned(10) (=A)
    81                              # array(2)
       19 03e7                      # unsigned(999) (=cose+cbor)
```

Figure 5: SAM Information Payload Example encoded in CBOR

3.3. Access Request

To retrieve an access ticket for the resource that C wants to access, C sends an Access Request to its CAM. The Access Request is constructed as follows:

1. The request method is POST.

2. The request URI is set as described below.

3. The message payload contains a COSE_encryptData or COSE_enveloped structure with content-type application/dcaf+cbor that describes the action and resource for which C requests an access ticket.

The request URI identifies a resource at CAM for handling authorization requests from C. The URI SHOULD be announced by CAM in its resource directory as described in [I-D.gerdes-ace-dcaf-authorize].

Note: Where capacity limitations of C do not allow for resource directory lookups, the request URI in Access Requests could be hard-coded during provisioning or set in a specific device configuration profile.

The message payload is constructed from the SAM information that S has returned as described in [I-D.gerdes-ace-dcaf-authorize]. An example Access Request from C to CAM is depicted in Figure 6. (Refer to [I-D.gerdes-ace-dcaf-authorize] for a detailed description of the available attributes and their semantics.)

```
POST /client-authorize
Content-Format: application/cose+cbor
[ h'a1031862',      # protected { content_type: application/dcaf+cbor }
  { alg: AES-CCM-16-64-128         # unprotected
    nonce: h'd6150b90e6f0eb5be42164062c', # nonce
  },
  h'TBD:encrypted payload w/ tag', # encrypted DCAF payload
  # recipients:
  [ [ h'',                         # protected (absent for AE algorithm)
      { alg: A128KW,               # unprotected
        kid: h'383261622e6161733432' # context identifier: "82ab.aas42"
      },
      h'52ff9ed52d...'             # encrypted session key
  ] ]
]
```

Figure 6: Access Request Message Example

The example shows an Access Request message with COSE payload that contains the encrypted and integrity protected DCAF object shown in Figure 7. To integrity-protect the CoAP message header fields the canonicalized CoAP message MUST be included in the external_aad structure. The recipient structure of this message contains a wrapped key that is encrypted with the key material for the common security context of C and CAM that is identified by the kid parameter. If the client cannot create a random session key, it

could send a COSE_encryptData structure instead using the direct encryption method. The benefit of wrapping the content encryption key is that CAM can pass the encrypted content on to SAM needing to wrap the content encryption key with the key material used in the common security context with SAM.

```
{
  SAM: "coaps://sam.example.com/authorize",
  SAI: ["coaps://temp451.example.com/s/tempC", 5],
  TS: 168537
}
```

Figure 7: Access Request Payload Example

The example shows an Access Request message for the resource "/s/tempC" on the Server "temp451.example.com". Requested operations in attribute SAI are GET and PUT.

The attributes SAM (that denotes the Server Authorization Manager to use) and TS (a nonce generated by S) are taken from the SAM Information message from S.

The response to an Authorization Request is delivered by CAM back to C in a Ticket Transfer message.

3.4. Ticket Request Message

CAM processes any Access Request message received from C as defined in [I-D.gerdes-ace-dcaf-authorize]. If CAM decides to send a Ticket Request message to the SAM provided in the Access Request, it has to establish a security context with SAM. Depending on the URI scheme used in the SAM field of the Access Request message payload (the less-constrained devices CAM and SAM do not necessarily use CoAP to communicate with each other), this could be, e.g., a DTLS channel (for "coaps") or a TLS connection (for "https"), or a COSE_enveloped structure using SAM’s public key to encrypt the content encryption key.

3.5. Ticket Grant Message

A Ticket Request Message is processed and responded to as specified in [I-D.gerdes-ace-dcaf-authorize]. SAM MUST use the same security context that has been used by CAM to transfer the Ticket Request message, i.e., if the Ticket Request message was received over DTLS, the response MUST be sent over the same DTLS session. This restriction is alleviated slightly when using COSE where the only requirement is that the CoAP response can be mapped to the respective request.

3.6. Ticket Transfer Message

A Ticket Transfer message is sent by CAM to deliver the authorization information from SAM in a Ticket Grant message to the requesting client C. Processing of the Ticket Grant message and construction of the Ticket Transfer message is done as specified in [I-D.gerdes-ace-dcaf-authorize]. An example for a Ticket Transfer message in response to the Ticket Access Request described in Section 3.3 is depicted in Figure 8.

```
2.05 Content
Content-Format: application/cose+cbor
[ h'a1031862',     # protected { content_type: application/dcaf+cbor }
  { alg: AES-CCM-16-64-128         # unprotected
    nonce: h'd259f53783993e757ec9d1d957', # nonce
    kid: h'383261622e6161733432'   # context identifier: "82ab.aas42"
  },
  h'TBD:encrypted payload w/ tag', # encrypted DCAF payload
]
```

Figure 8: Example Ticket Transfer Message Encoded as COSE Message

In this example, a COSE_encryptData structure is used to avoid including a recipients structure. The kid parameter referring to the same security context that has been used for the Access Request message is included with the unprotected header of the COSE_encryptData structure. The encrypted DCAF payload contains the required ticket Face and Verifier as defined in [I-D.gerdes-ace-dcaf-authorize]. In this example, the ticket shown in Figure 9 is passed in the payload field of the COSE_encryptData structure shown in Figure 8.

```
{ F: {
        SAI: [ "/s/tempC", 7 ],
        TS: 0("2013-07-10T10:04:12.391"),
        L:  86400,
        G: hmac_sha256
  },
  V: h'f89947160c73601c7a65cb5e08812026
       6d0f0565160e3ff7d3907441cdf44cc9'
  CAI: [ "/s/tempC", 1 ],
  TS: 0("2013-07-10T10:04:12.855"),
  L:  86400
}
```

Figure 9: Example Ticket Transfer Message

3.7. Security Association between C and S

The information contained in a Ticket Transfer message (i.e. a ticket a Face and Client Information) can be used by C to establish a security context with S. While [I-D.gerdes-ace-dcaf-authorize] defines how to infer a DTLS pre-shared key, this specification uses the verifier as MAC key in a COSE_MAC structure as described below. This structure comprises the payload of a POST request to the authorization resource hosted by S as described in Section 7.

1. The CoAP request is protected as external_aad as described in Section 6.

2. The protected header contains the parameter content_type with the value 'application/dcaf+cbor'.

3. The unprotected header contains the alg parameter that denotes the MAC algorithm that is used at the content level.

4. The payload field of the COSE_MAC structure contains the ticket Face encoded as canonicalized CBOR structure, and the tag field is constructed using the verifier from the Ticket Transfer message as secret, and the recipients structure is filled with empty values.

The authorization for uploading authorization tickets is tied to a key that is associated to the particular ticket Face and MUST be generated by the authorized SAM. When receiving a POST request to the auth-info resource, S generates its own version of the verifier using the information contained in Face.

The distributed key derivation method is defined as follows:

- SAM and S both generate the verifier using the information included in Face. They use an HMAC algorithm on Face with a shared key K(SAM,S). The result serves as the content encryption key. How SAM and S exchange K(SAM,S) is not in the scope of this document. They MAY use their preshared key as K(SAM,S).

- SAM MUST include a representation of the session key in the Verifier.

- As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using protected channels.

- SAM MUST NOT include a representation of the Verifier in Face.

- SAM MUST NOT encrypt Face.

Once S has validated the contents of the POST request using the locally generated verifier, it creates a new resource that represents this authorization and returns the Location-Path of this new resource. This path then can be used by C to update the authorization information and MUST be used by C in the kid parameter to identify this security context as described in Section 2.1.

An example for the POST request and corresponding 2.01 response is given in Figure 10. The Location-Path returned by S is subsequently used by C as identifier for the security context tied to this authorization.

```
C --> S
POST /authorize
Content-Format: application/cose+cbor
[ h'a1031862',     # protected { content_type: application/dcaf+cbor }
  { alg: HMAC 256/256 },        # unprotected
  h'{ SAI: [ "/s/tempC" ... }', # DCAF payload wrapped in CBOR binary
  h'....',              # tag: HMAC(options+protected+payload, secret)
  [ [ h'', {}, h'' ] ]          # recipients
]

S --> C
2.01 Created
Content-Format: application/cose+cbor
Location-Path: 238dsa29
Authorization: [ h'a1031862',   # protected
  { alg: HMAC 256/256 },        # unprotected
  h'',                          # empty payload
  h'....',                      # tag: HMAC(options+protected, secret)
  [ [ h'', {}, h'' ] ]          # recipients
]
```

Figure 10: Example POST to S's auth-info Resource and Response

4. Piggybacked Protected Content

Piggybacked protected content was introduced in [I-D.gerdes-ace-dcaf-authorize] as a possibility to deliver an encrypted resource representation without having to maintain authorization information for the respective resource. Once a requesting client has received the piggybacked content, it needs to request authorization for accessing the protected data. To do so, it constructs an Access Request as defined in Section 3.3. If access to the protected data is granted, the requesting client will be provided with cryptographic material to verify the integrity and authenticity of the piggybacked content and decrypt the protected data in case it is encrypted.

5. CoAP Options Authorization and Authorization-Format

The options Authorization and Authorization-Format have the properties shown in Table 1.

No.	C	U	N	R	Name	Format	Length	Default
64					Authorization	opaque	1-1034	(none)
65	x				Authorization-Format	uint	0-2	application/cose+cbor

Table 1: The Options Authorization and Authorization-Format

6. Canonicalization of the CoAP Message Header

This section describes the canonicalization of parts from the CoAP message for integrity protection. As intermediaries such as caching proxies may change certain fields in a CoAP message, only those fields are considered that must not be changed by intermediaries. The canonicalized CoAP message then serves as external_aad to the COSE MAC_structure and Enc_structure as used in this specification. The canonicalized CoAP message is constructed as follows:

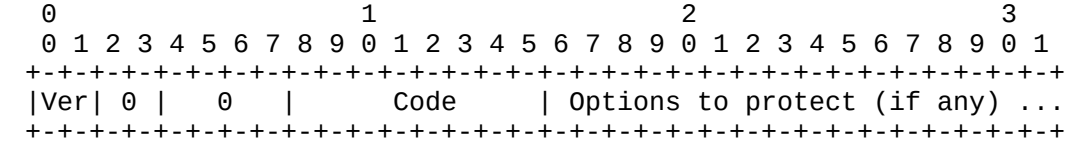


Figure 11: Canonicalized CoAP Message Header

As shown in Figure 11, only the version bits and the message code from the CoAP base header are relevant for integrity protection. [_3] From the list of options that a message might have, only the following options are to be included with the canonicalized message. [_4]

- If-Match
- Uri-Host
- ETag

- If-None-Match
- Observe
- Uri-Port
- Location-Path
- Uri-Path
- Uri-Query
- Accept
- Location-Query
- Proxy-Uri
- Proxy-Scheme
- Size1

Note: The Content-Format must be contained in the protected header of the MAC_structure or Enc_structure and hence is not required here.

An application that requires integrity protection of new options that are not listed here must add a critical-options header field to the MAC_structure or Enc_structure containing a CBOR array with the additional options to protect in ascending numerical order.

Figure 12 shows an example for a POST request to upload SenML [I-D.jennings-core-senml] sensor readings to a remote server. The protected header in the COSE_Mac structure contains a 'required options' entry that lists the custom option X-Something, hence the external_aad would contain a canonicalized message header that consists of the CoAP version number, the method POST, Uri-Path 'measurements', Uri-Path 'current', and X-Something 1234 as delta-encoded options in ascending order as specified in Section 3.1 of [RFC7252].

```
POST /measurements/current
Content-Format: application/senml+cbor
X-Something: 1234
Authorization: [
  # protected { content_type: application/senml+cbor,
  #             "required options": [ X-Something ] }
  h'a203766170706c69636174696f6e2f73656e6d6...',
  { alg: HMAC 256/256,          # unprotected
    kid: h'3233386473613239'    # context identifier: "238dsa29"
  },
  h'a2231a4eaecc5d....',        # payload: "{ -4: 1320078..."
  h'....',            # tag: HMAC(options+protected+payload, secret)
  [ [ nil, {}, h'' ] ]          # recipients
]

{ -4: 1320078429,
  -2: [{0: "temperature", 2: 272, 1: "Cel"},
       {0: "humidity", 2: 80, 1: "%RH"}]
}
```

Figure 12: Example Message with Protected Custom Option

7. The "auth-info" Link Relation

This section defines a resource type "auth-info" that can be used by clients to establish a new security context with S using the authorization information retrieved from SAM. When used with the parameter rt in a web link, "auth-info" indicates that the corresponding target URI can be used in a POST message to upload the authorization information contained in the request payload.

The following example shows the web link used by S in this document to accept authorization information created by SAM.

```
<authorize>;rt="auth-info";ct=TBD1,ct_cose_msg
           ;title="Upload Authorization Information"
```

The resource directory that hosts the resource descriptions of S could list the following description. In this example, the URI "ep/node138/a/switch2941" is relative to the resource context "coaps://sam.example.com/", i.e. the Server Authorization Manager SAM.

```
<ep/node138/a/switch2941>;rt="auth-info";ct=TBD1,ct_cose_msg
                         ;ep="node138"
                         ;title="Upload Authorization Information"
                         ;anchor="coaps://s.example.com/"
```

8. Security Considerations

The SAM Information message cannot be protected as no security context between S and C is present at the time the message is sent. An attacker thus can inject a SAM Information message listing a different SAM URI to trick C into disclosing the intended action. Where this is an issue, C could retrieve the SAM URI from a resource directory as described in [I-D.gerdes-ace-dcaf-authorize].

9. IANA Considerations

The following registrations are done following the procedure specified in [RFC6838].

Note to RFC Editor: Please replace all occurrences of "[RFC-XXXX]" with the RFC number of this specification.

9.1. CoAP Option Registration

IANA is requested to add the following entries to the CoAP Option Numbers registry:

Number	Name	Reference
64	Authorization	[RFC-XXXX]
65	Authorization-Format	[RFC-XXXX]

10. Acknowledgements

The authors would like to thank Carsten Bormann for his valuable input and feedback.

11. References

11.1. Normative References

[I-D.gerdes-ace-dcaf-authorize]
Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-03 (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>.

[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>.

11.2. Informative References

[I-D.ietf-ace-actors] Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-02 (work in progress), October 2015.

[I-D.ietf-cose-msg] Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-06 (work in progress), October 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-02 (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>.

[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>.

Editorial Comments

[_1] Editor’s note: As a consequence, if no such security context is found, the request will be rejected as Unauthorized Request.

[_3] Editor’s note: message type, token and id can change on the way.

[_4] TBD

Authors' Addresses

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63904
Email: bergmann@tzi.org

Stefanie Gerdes
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63906
Email: gerdes@tzi.org

CoRE
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

C. Bormann
Universitaet Bremen TZI

October 19, 2015

CoAP FETCH Method
draft-bormann-core-coap-fetch-00

Abstract

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.

The present proposal adds a new CoAP method, FETCH, to perform the equivalent of a GET with a request body.

This 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. For now, we therefore prefer to discuss the proposal under a different name, for which we have chosen the GET synonym FETCH.

Status of This Memo

Table of Contents

1. Introduction 2
2. FETCH 3
2.1. The Content-Format Option 4
2.2. Working with Observe 4
2.3. Working with Block 5
2.4. Discussion 5
3. Security Considerations 5
4. IANA Considerations 5
5. Acknowledgements 5
6. References 5
6.1. Normative References 5
6.2. Informative References 6
Author's Address 6

1. Introduction

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 solely to enable supplying a larger set of parameters to the search than can be comfortably transferred in the URI with a GET request. [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.

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:

- 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.

- 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.

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].

2. FETCH

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 based on the request parameters (described by the request options and 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.

The body of the request 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 the 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 expected 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 the 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 payload 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 request 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. [[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. 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. Security Considerations

The FETCH method is subject to the same general security considerations as all CoAP methods as described in [RFC7252].

4. IANA Considerations

IANA is requested to add an entry to the sub-registry "CoAP Method Codes":

Code	Name	Reference
0.07	FETCH	[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.

5. Acknowledgements

Most of the text in this I-D was stolen, e.g. from [I-D.snell-search-method] or from [I-D.vanderstok-core-patch]. Thank you!

Klaus Hartke found a number of problems while quickly checking an earlier version of this document.

6. References

6.1. 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>.

6.2. Informative References

[I-D.hartke-core-apps] Hartke, K., "CoRE Application Descriptions", draft-hartke-core-apps-02 (work in progress), August 2015.

[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.

[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.

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

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:

```
RTO := 0.25 * E_weak_ + 0.75 * RTO (1)

RTO := 0.5 * E_strong_ + 0.5 * RTO (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 x RTO and ACK_RANDOM_FACTOR x 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

RTO := 1 s + (0.5 * 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/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/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.

PLIMIT = max(LAMBDA, LAMBDA*ACK_TIMEOUT)/mean(RTO)) (3)

PLIMIT = LAMBDA (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:

PLIMIT= max(4, (4*2 s)/mean(0.5 s, 1.5 s)) = max (4, 8 s/1 s) = max (4, 8) = 8

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 Beroset, 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
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

Copyright Notice

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:

- reducing implementation complexity in order to minimize code size,
- 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,
- 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:

- Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- The transfer of each block is acknowledged, enabling individual retransmission if required.
- Both sides have a say in the block size that actually will be used.
- The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- 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:

- by the maximum datagram size (~ 64 KiB for UDP)
- by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- 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:

- The size of the block (SZX);
- 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.)

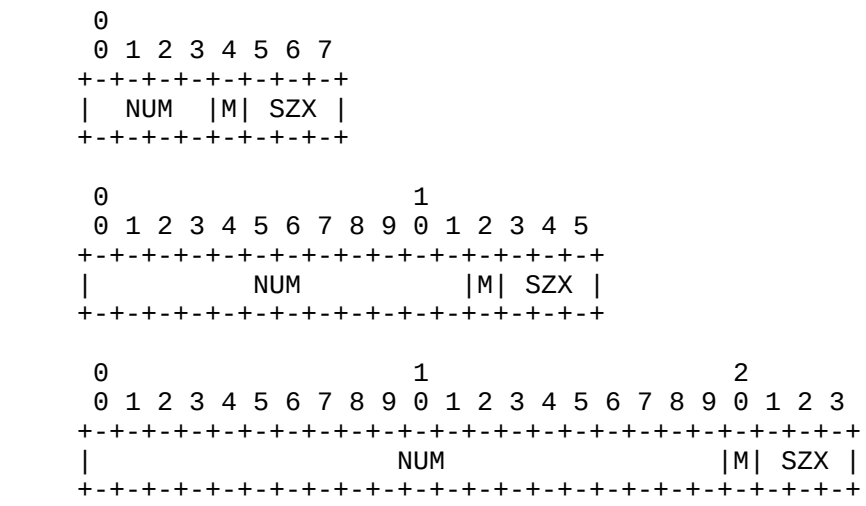


Figure 1: Block option value

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**(SZX + 4)". SZX is transferred in the three least significant bits of the option value (i.e., "val & 7" where "val" is the value of the option).

The fourth least significant bit, the M or "more" bit ("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 "NUM << (SZX + 4)".

Implementation note: As an implementation convenience, "(val & ~0xF) << (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:

- 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.)

- 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).

- 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.

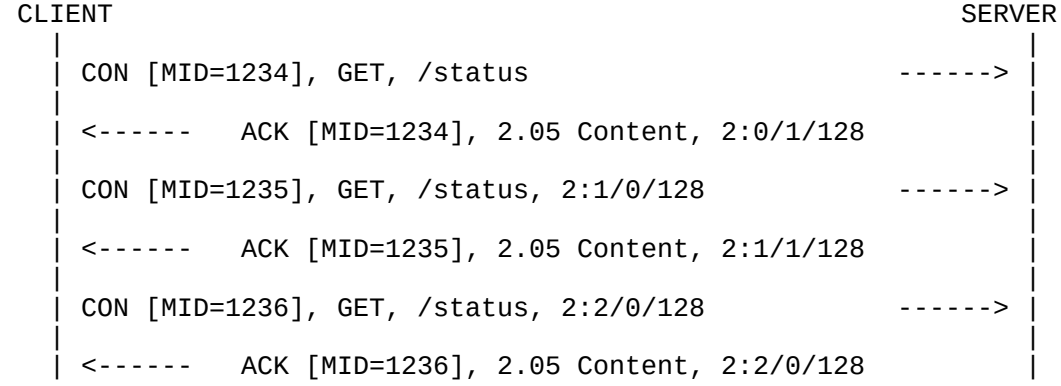


Figure 2: Simple block-wise GET

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.

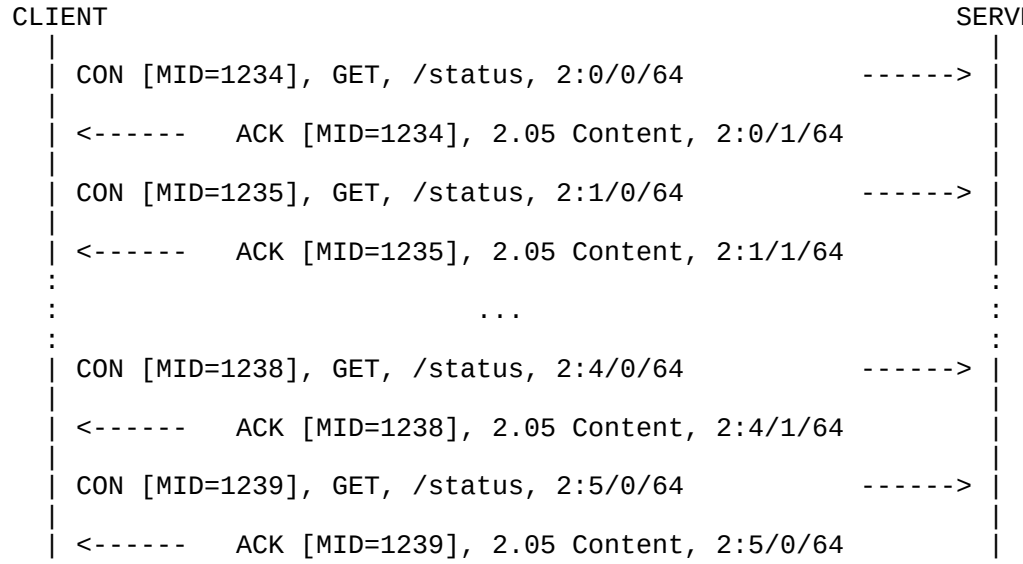


Figure 3: Block-wise GET with early negotiation

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.

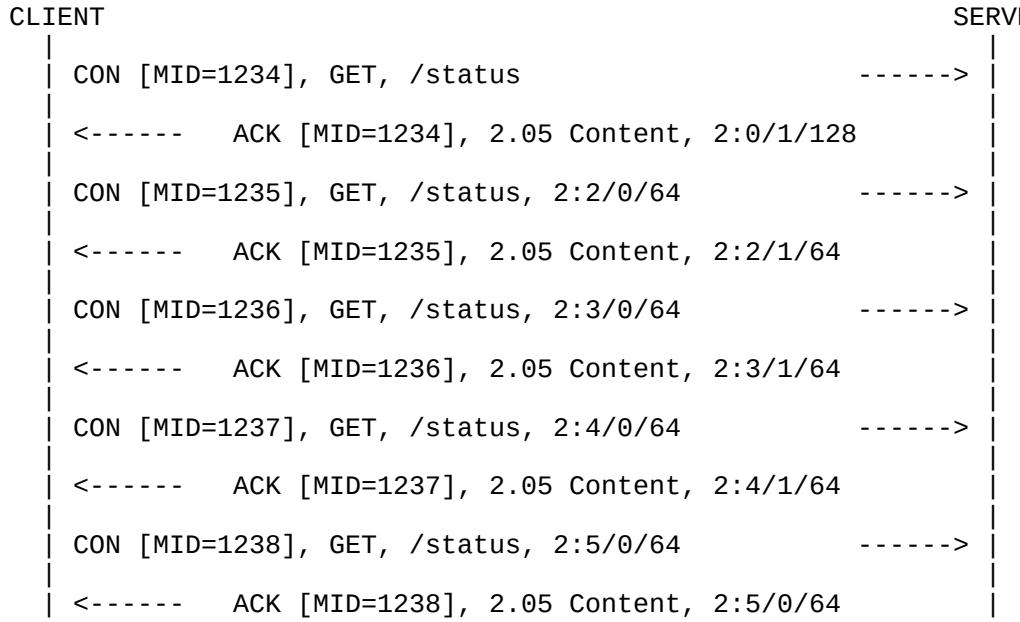


Figure 4: Block-wise GET with late negotiation

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).

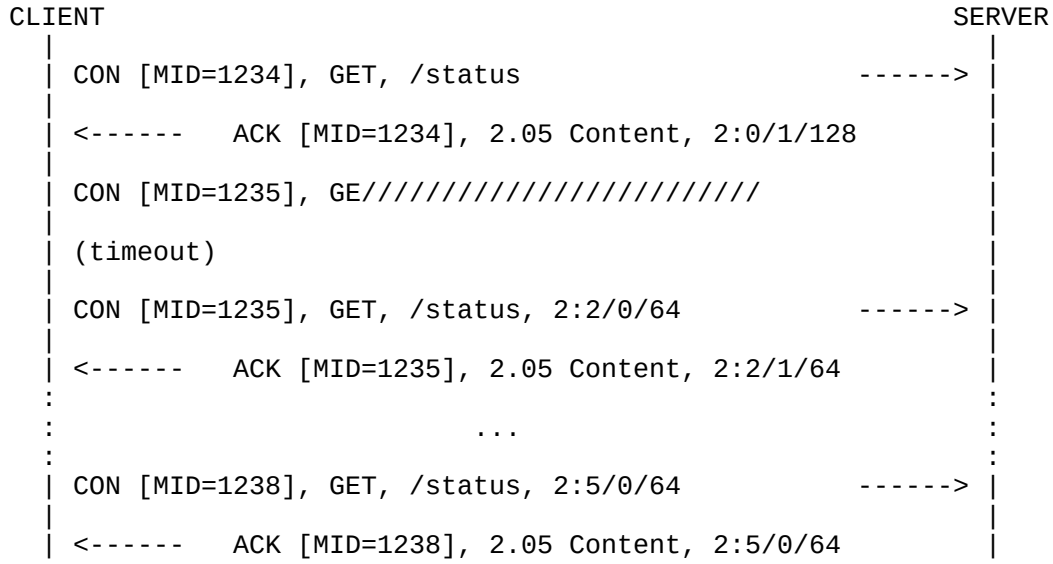


Figure 5: Block-wise GET with late negotiation and lost CON

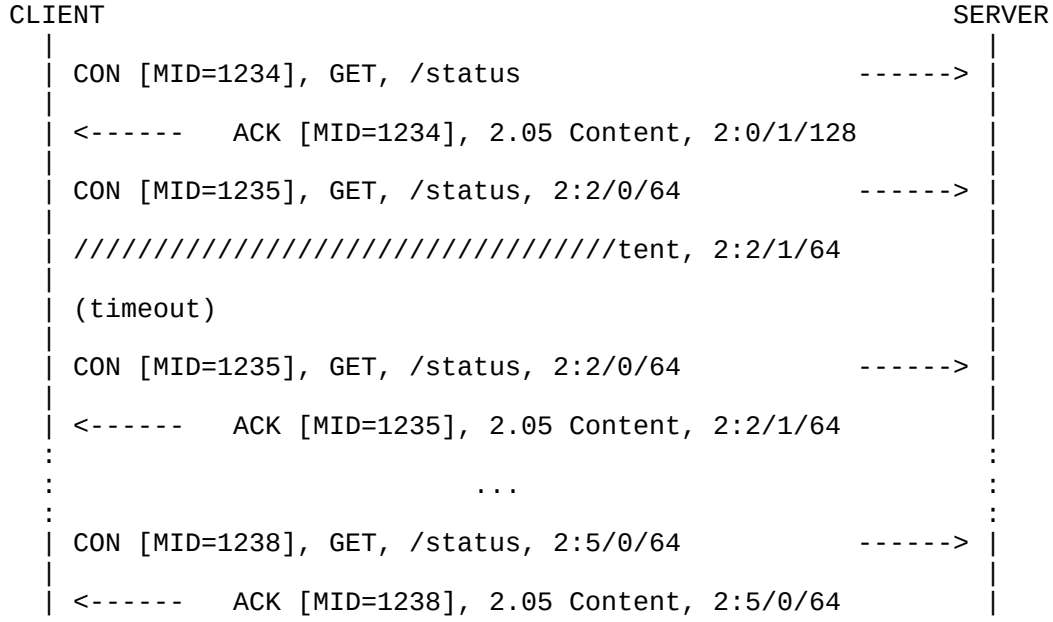


Figure 6: Block-wise GET with late negotiation and lost ACK

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.

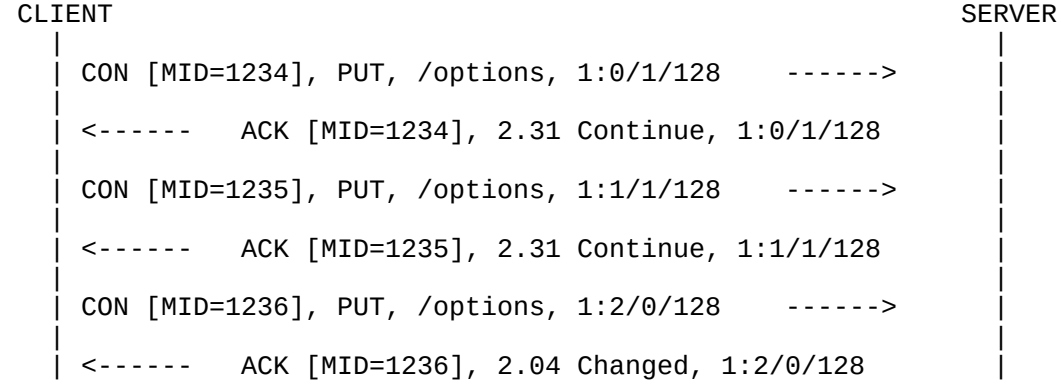


Figure 7: Simple atomic block-wise PUT

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.

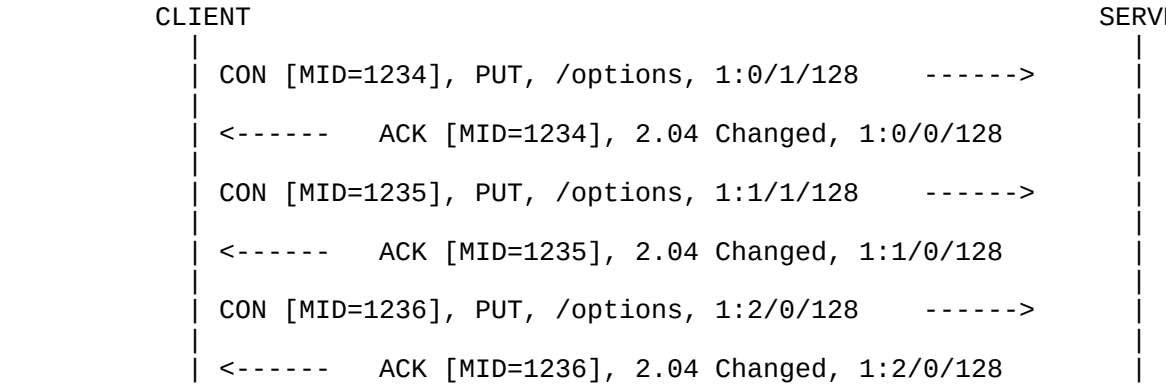


Figure 8: Simple stateless block-wise PUT

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.

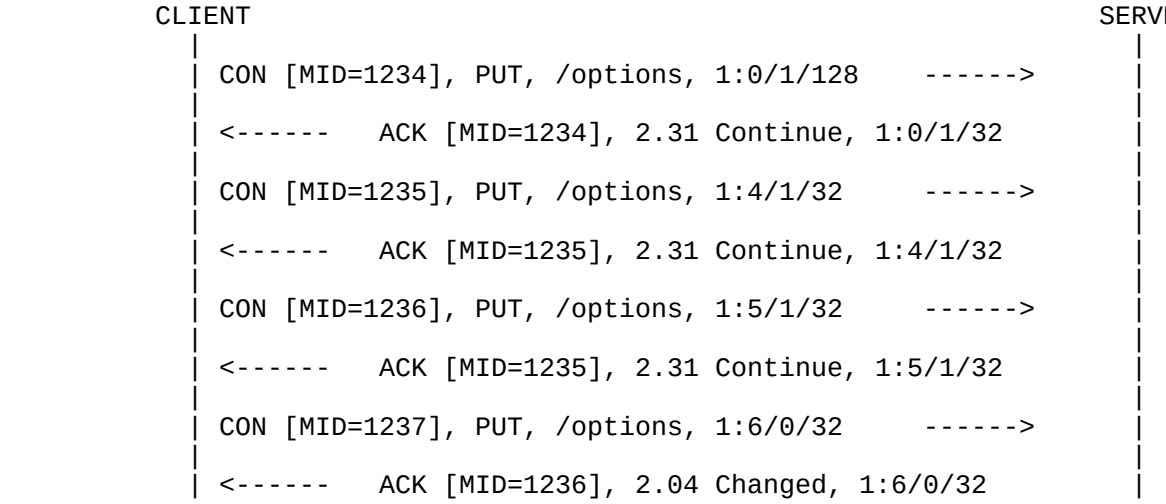


Figure 9: Simple atomic block-wise PUT with negotiation

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.

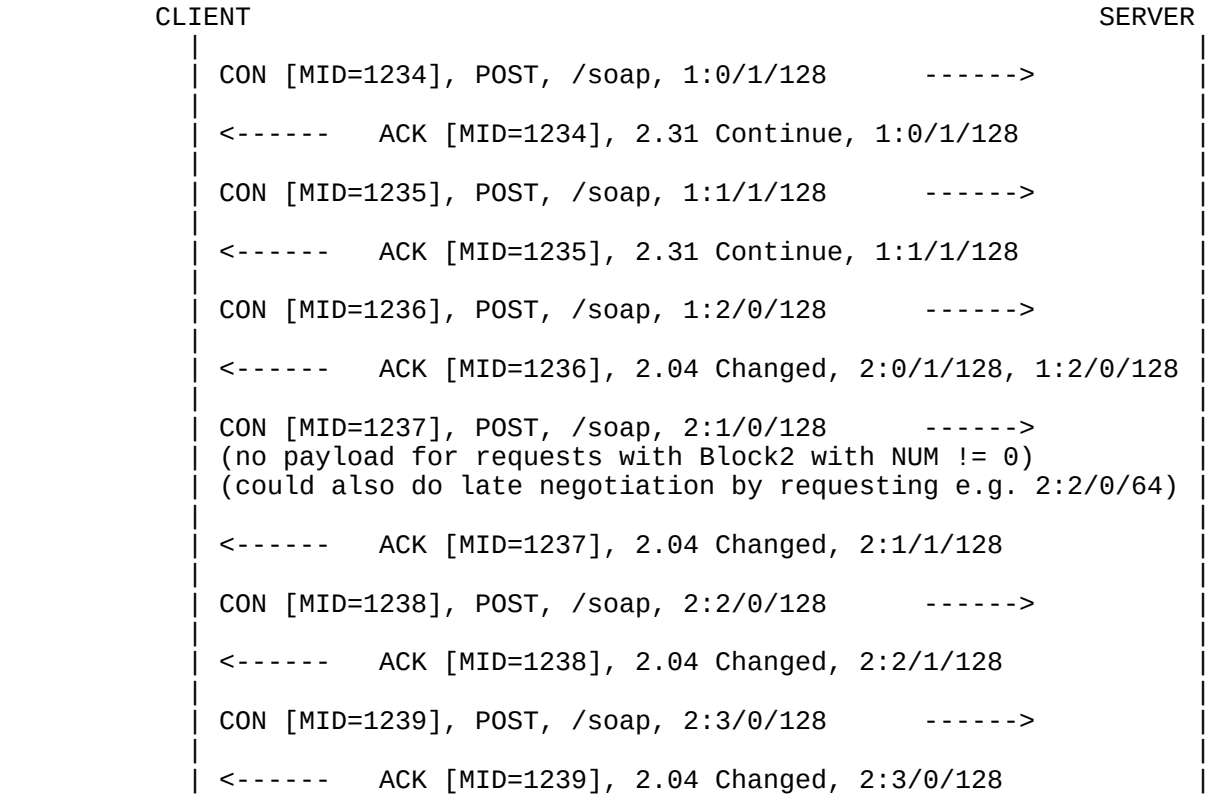


Figure 10: Atomic block-wise POST with block-wise response

This model does provide for early negotiation input to the Block2 block-wise transfer, as shown below.

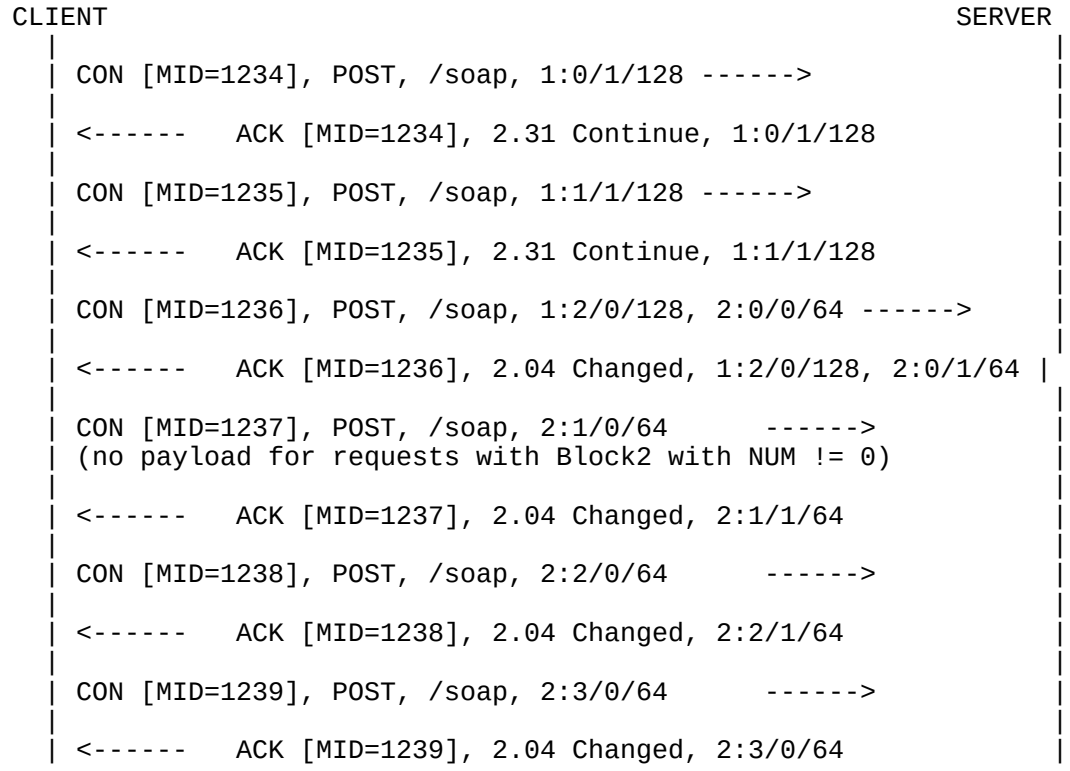
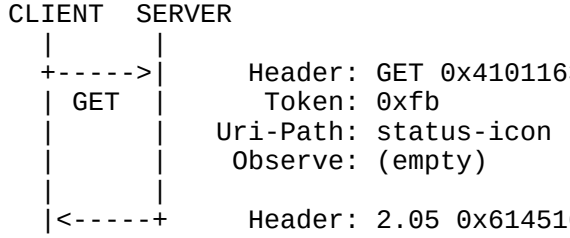


Figure 11: Atomic block-wise POST with block-wise response, early negotiation

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.



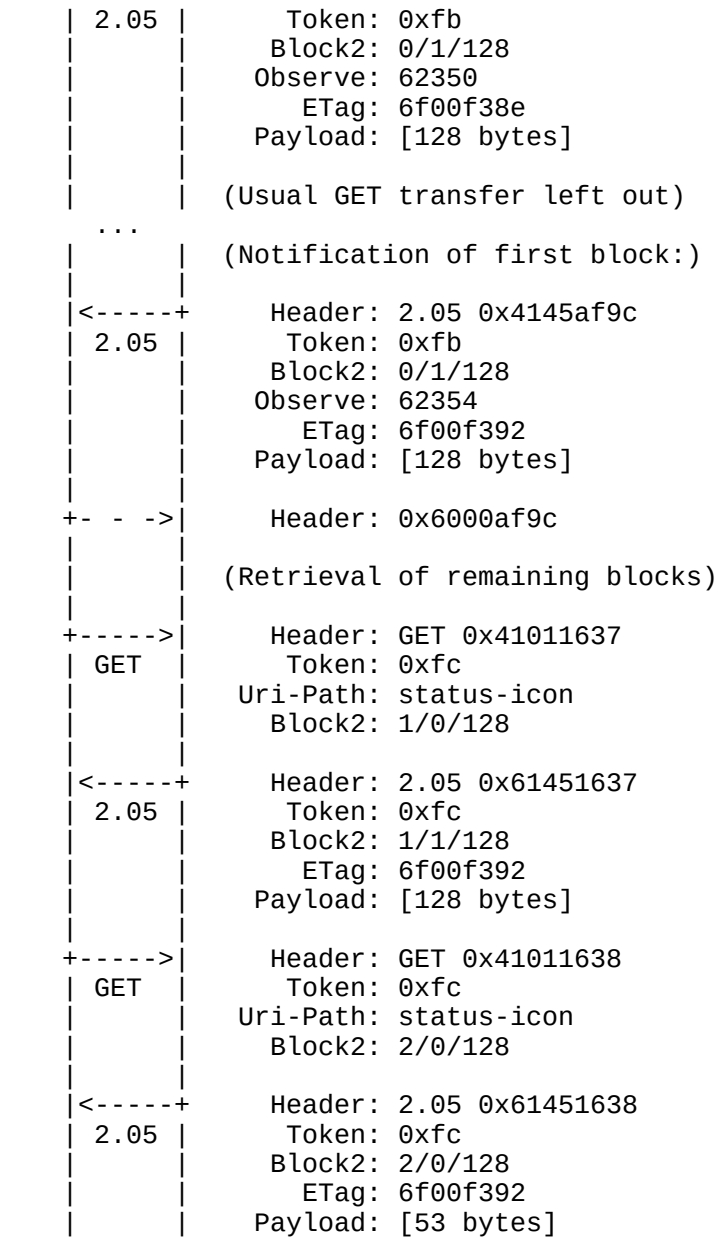
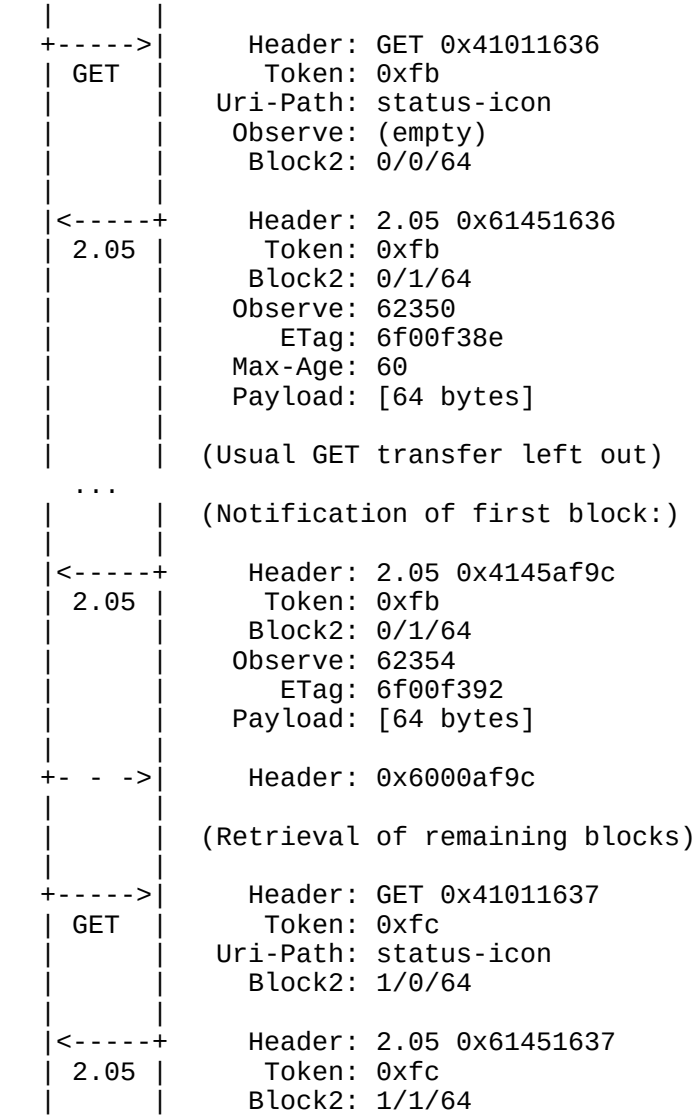


Figure 12: Observe sequence with block-wise response

(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.



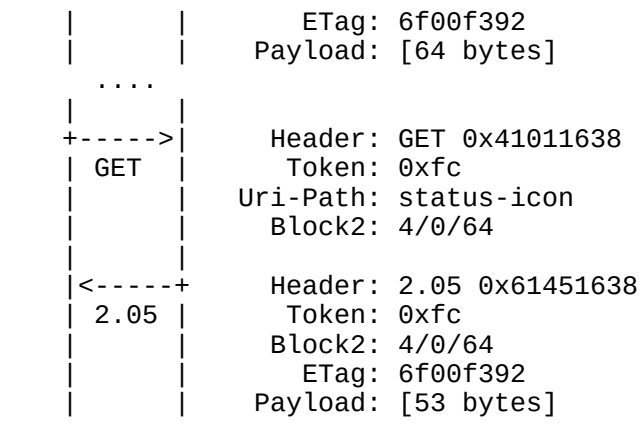


Figure 13: Observe sequence with early negotiation

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:

- 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.

- 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:

- 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").

- 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:

- 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.

- The end of a block-wise transfer is governed by the M bits in the Block Options, _not_ by exhausting the size estimates exchanged.

- 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.

- 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 Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2016

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
Alcatel-Lucent
E. Dijk
Philips Research

July 3, 2015

Guidelines for HTTP-CoAP Mapping Implementations
draft-ietf-core-http-mapping-07

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.

Table of Contents

1. Introduction 3
2. Terminology 4
3. HTTP-CoAP Reverse 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 8
5.2.2. Encoding Caveats 9
5.3. URI Mapping Template 9
5.3.1. Simple Form 9
5.3.2. Enhanced Form 11
5.4. Discovery 13
5.4.1. Discovering CoAP Resources 13
5.4.2. Examples 14
6. Media Type Mapping 15
6.1. Overview 15
6.2. 'application/coap-payload' Media Type 17
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 20
8. Additional Mapping Guidelines 23
8.1. Caching and Congestion Control 23
8.2. Cache Refresh via Observe 23
8.3. Use of CoAP Blockwise Transfer 24
8.4. Security Translation 25
8.5. CoAP Multicast 25
8.6. Timeouts 26
8.7. Miscellaneous 26
9. IANA Considerations 26
9.1. New 'core.hc' Resource Type 26
9.2. New 'coap-payload' Internet Media Type 27

10. Security Considerations 28
10.1. Traffic Overflow 29
10.2. Handling Secured Exchanges 29
10.3. Proxy and CoAP Server Resource Exhaustion 30
10.4. URI Mapping 30
11. Acknowledgements 31
12. References 31
12.1. Normative References 31
12.2. Informative References 32
Appendix A. Change Log 33
Authors' Addresses 35

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 REST architectures such as the Web. The latter goal has led to define 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, implementing such a cross-protocol proxy can be complex, and many details regarding its internal procedures and design choices require further elaboration. Therefore, a first goal of this document is to provide more detailed information to proxy designers and implementers, to help build proxies that correctly inter-work with existing CoAP and HTTP implementations.

The second goal of this informational document is to define a consistent set of guidelines that a HTTP-to-CoAP proxy implementation MAY adhere to. The main reason for adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability. (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.)

This document is organized as follows:

- Section 2 describes terminology to identify proxy types, mapping approaches and proxy deployments;
- Section 3 introduces the reverse HTTP-CoAP proxy;
- Section 4 lists use cases in which HTTP clients need to contact CoAP servers;

- Section 5 introduces a default HTTP-to-CoAP URI mapping syntax;
- Section 6 describes how to map HTTP media types to CoAP content formats and vice versa;
- Section 7 describes how to map CoAP responses to HTTP responses;
- Section 8 describes additional mapping guidelines related to caching, congestion, timeouts and CoAP blockwise [I-D.ietf-core-block] transfers;
- 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: 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: 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 [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. HTTP-CoAP Reverse Proxy

A Reverse HTTP-CoAP Proxy (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 an HC Proxy is placed server-side, at the boundary of the Constrained Network domain, to avoid any HTTP traffic on the Constrained Network and to avoid any (unsecured) CoAP multicast traffic outside the Constrained Network. The DNS server is used by the HTTP Client to resolve the IP address of the HC Proxy and optionally also by the HC Proxy to resolve IP addresses of CoAP servers.

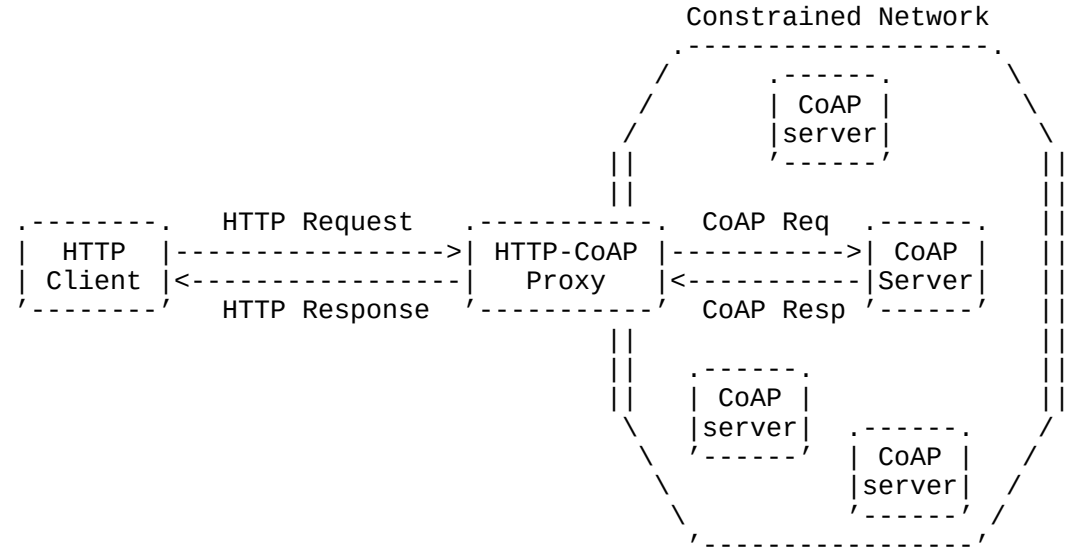


Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

Other placement options for the HC Proxy (not shown) are client-side, 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, that section only considers the case of a Forward HC Proxy in which a client explicitly indicates it targets a request to a CoAP server, and does not cover all aspects of proxy implementation in detail. 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. Smartphone and home sensor: A smartphone can access directly a CoAP home sensor using an authenticated 'https' request, if its home router contains an HC proxy. An HTML5 application on the smartphone can provide a friendly UI to the user using standard (HTTP) networking functions of HTML5.

2. 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 an HC proxy.

3. Making sensor data available to 3rd parties: For demonstration or public interest purposes, a 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 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.

5. URI Mapping

Though, in principle, a CoAP URI could be directly used by a HTTP user agent 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 "pack" a CoAP URI into a HTTP URI so that it can be (non-destructively) transported from the user agent 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 process through which the URI of a CoAP resource is transformed into an HTTP URI so that:

- the requesting HTTP user agent can handle it;
- the receiving HC proxy can extract the intended CoAP URI unambiguously.

To this end, the remainder of this section will identify:

- the default mechanism to map a CoAP URI into a HTTP URI;
- the URI template format to express a class of CoAP-HTTP URI mapping functions;
- the discovery mechanism based on CoRE Link Format [RFC6690] through which clients of an HC proxy can dynamically discover information about the supported URI Mapping Template(s), as well as the base 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:

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 query) component(s) embed the information used by an 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 a base URI provided by the HC proxy, to form the Hosting HTTP URI.

For example: given a base 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 base 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 an 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 an HC proxy to inform its clients about the expected syntax for the Hosting HTTP URI.

When instantiated, an URI Mapping Template is always concatenated to a base URI provided by the 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 base 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 base 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 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 is associated with a base URI, and can be used as the value for the "rt" attribute in a query to the /.well-known/core in order to locate the base URI where the HC proxy function is anchored.

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 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. Discovering CoAP Resources

For a HTTP client, it may be unknown which CoAP resources are available through a HC Proxy. By default an HC Proxy does not support a method to discover all CoAP resources. However, if an HC Proxy is integrated with a Resource Directory ([I-D.ietf-core-resource-directory]) function, an HTTP client can

discover all CoAP resources of its interest by doing an RD Lookup to the HC Proxy, via HTTP. This is possible because a single RD can support both CoAP and HTTP interfaces simultaneously. Of course the HTTP client will this way only discover resources that have been previously registered onto this RD by CoAP devices.

5.4.2. Examples

- The first example exercises the CoAP interface, and assumes that the default template, {+tu}, is used:

```
Req:  GET coap://[ff02::1]/.well-known/core?rt=core.hc

Res:  2.05 Content
      </hc>;anchor="http://p.example.com";rt="core.hc"
```

- 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="http://p.example.com";
      rt="core.hc";hct="?uri={+tu}"
```

On the HTTP side, link information can be serialized in more than one way:

- 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"
```

- using the 'application/link-format+json' content type as defined in [I-D.bormann-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"}]
```

- 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"
```

- An HC proxy may expose two different base 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

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 side, when the 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 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", would '60' be an unknown content format to the 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 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.

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:

- 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.
- If C-E is not given it is assumed to be "identity".
- 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 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 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 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 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]. In this scenario, the CoAP response diagnostic payload MUST NOT be returned as the regular HTTP payload (message body). Instead, the CoAP diagnostic payload must be used as the HTTP reason-phrase of the HTTP status line, as defined in Section 3.1.2 of [RFC7230], without any alterations, except those needed to comply to the reason-phrase ABNF definition.

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	401 Unauthorized	5
4.02 Bad Option	400 Bad Request	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 HC proxy, for efficiency reasons, is caching resources and translated a HTTP request (without conditional request) to a CoAP request that includes ETag validation. The proxy receiving 2.03 updates the freshness of its cached representation and returns the entire representation to the HTTP client.

5. A HTTP 401 Unauthorized (Section 3.1 of [RFC7235]) response MUST include a WWW-Authenticate header. Since there is no CoAP equivalent of WWW-Authenticate, the HC proxy must generate this header itself including at least one challenge (Section 4.1 of [RFC7235]). If the HC proxy does not implement a proper authentication method that can be used to gain access to the target CoAP resource, it can include a 'dummy' challenge for example "WWW-Authenticate: None".

6. A proxy receiving 4.02 may first retry the request with less CoAP Options in the hope that the CoAP server will understand the newly formulated request. For example, if the proxy tried using a Block Option [I-D.ietf-core-block] which was not recognized by the CoAP server it may retry without that Block Option. Note that HTTP 402 MUST NOT be returned because it is reserved for future use [RFC7231].

7. A CoAP 4.05 (Method Not Allowed) response SHOULD normally be mapped to a HTTP 400 (Method Not Allowed) 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 "405" 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.

8. Additional Mapping Guidelines

8.1. Caching and Congestion Control

An HC proxy SHOULD limit the number of requests to CoAP servers by responding, where applicable, with a cached representation of the resource.

Duplicate idempotent pending requests by an HC proxy to the same CoAP resource SHOULD in general be avoided, by using the same response for multiple requesting HTTP clients without duplicating the CoAP request.

If the HTTP client times out and drops the HTTP session to the HC proxy (closing the TCP 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 interactions to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the 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 HC proxy should be server-side placed.

Resources experiencing a high access rate coupled with high volatility MAY be observed [I-D.ietf-core-observe] by the 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 [I-D.ietf-core-observe] 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*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

An 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.

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:

- calculated based on a known or typical UDP datagram buffer size for CoAP end-points, or
- set to N times the known size of a link-layer frame in a constrained network where e.g. N=5, or
- preset to a known IP MTU value, or
- 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 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 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 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 an 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

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; 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 HC proxy SHOULD respond 403 (Forbidden).

If an 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, 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.

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].

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 may use confirmable CoAP requests, such timeout value T SHOULD be at least

T = MAX_RTT + 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 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 an HC proxy.

HTTP connection pipelining (section 6.3.2 of [RFC7230]) may be supported by an HC proxy. This is transparent to the CoAP servers: the HC proxy will serve the pipelined requests by issuing different CoAP requests. The 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 HC proxy scenario. In fact, the HC proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the HC proxy cannot be dropped, because the protocol translation function is the core duty of the 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 an 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 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 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 an HC proxy.

If a policy for access to ’coaps’ URIs is configurable in an 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, an 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 SS HC proxy deployment shown in Figure 1), the 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 HC proxy, i.e., the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The 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 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 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. An 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-17 (work in progress), March 2015.

[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, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7232] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, June 2014.

[RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

12.2. Informative References

[I-D.bormann-core-links-json]
Bormann, C., "Representing CoRE Link Collections in JSON", draft-bormann-core-links-json-02 (work in progress), February 2013.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-03 (work in progress), June 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, June 1999.

[RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.

[RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

[RFC7390] Rahman, A. and E. Dijk, "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, October 2014.

Appendix A. Change Log

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-06 to ietf-07:

- Addressed Ticket #384 - Section 5.4.1 describes briefly (informative) how to discover CoAP resources from an HTTP client.
- 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.
- 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.
- 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).
- 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:

- Fully restructured the draft, bringing introductory text more to the front and allocating main sections to each of the key topics; addressing Ticket #379;
- Addressed Ticket #382, fix of enhanced form URI template definition of q in Section 5.3.2;
- Addressed Ticket #381, found a mapping 4.01 to 401 Unauthorized in Section 7;
- Addressed Ticket #380 (Add IANA registration for "core.hc" Resource Type) in Section 9;
- 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;

- Removed details on the pros and cons of HC proxy placement options;
- Addressed review comments of Carsten Bormann;
- Clarified failure in mapping of HTTP Accept headers (Section 6.3);
- Clarified detection of CoAP servers not supporting blockwise (Section 8.3);
- Changed CoAP request timeout min value to MAX_RTT + MAX_SERVER_RESPONSE_DELAY (Section 8.6);
- Added security section item (Section 10.3) related to use of CoAP blockwise transfers;
- Many editorial improvements.

Changes from ietf-04 to ietf-05:

- 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);
- Addressed Ticket #375 (Add requirement on mapping of CoAP diagnostic payload) in Section 6.3.3.3 (Content Transcoding - Diagnostic Messages);
- Addressed comment from Yusuke (http://www.ietf.org/mail-archive/web/core/current/msg05491.html) in Section 6.3.3.1 (Content Transcoding - General);
- Various editorial improvements.

Changes from ietf-03 to ietf-04:

- Expanded use case descriptions in Section 4;
- Fixed/enhanced discovery examples in Section 5.4.1;
- 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);
- Updated HTTPBis WG draft references to recently published RFC numbers.
- Various editorial improvements.

Changes from ietf-02 to ietf-03:

- Closed Ticket #351 "Add security implications of proposed default HTTP-CoAP URI mapping";
- Closed Ticket #363 "Remove CoAP scheme in default HTTP-CoAP URI mapping";
- Closed Ticket #364 "Add discovery of HTTP-CoAP mapping resource(s)".

Changes from ietf-01 to ietf-02:

- Selection of single default URI mapping proposal as proposed to WG mailing list 2013-10-09.

Changes from ietf-00 to ietf-01:

- 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
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.

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 it’s 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 form 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 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 a 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 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 curent 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 Collection	core.hc	link-format, senml, 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 /l/ (Content-Format: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed

Req: GET /l/
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 /l/ (Content-Format: application/link-format)
</s/humidity>
Res: 2.04 Changed

Req: GET /l/ (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>

Req: GET /l/
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 /l/
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 Langauge (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 interfacee 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"/>
```

```
    <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,
     obervableRead, 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 an
d 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

- Fixed tickets #385 and #386
- Changed abstract and into to better describe content
- Focus on Interface and not function set/profiles in intro
- Changed references from draft-core-observe to RFC7641
- Moved Function sets and Profiles to section after Interfaces
- Moved Observe Attributes to the Link Binding section
- Add a Collection section to describe the collection types
- Add the Hypermedia Collection Interface Description

Changes from -02 to -03

- Added lt and gt to binding format section.
- Added pmin and pmax observe parameters to Observation Attributes
- Changed the definition of lt and gt to limit crossing.
- Added definitions for getattr and setattr to WADL.
- Added getattr and setattr to observable interfaces.

- Removed query parameters from Observe definition.
- Added observe-cancel definition to WADL and to observable interfaces.

Changes from -01 to -02

- Updated the date and version, fixed references.
- Removed pmin and pmax observe parameters [Ticket #336]

Changes from -00 to WG Document -01

- Improvements to the Function Set section.

Changes from -05 to WG Document -00

- Updated the date and version.

Changes from -04 to -05

- 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

- Draft refresh

Changes from -02 to -03

- Added Bindings
- Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- Defined a Function Set and its guidelines.
- Added the Link List interface.
- Added the Linked Batch interface.
- Improved the WADL interface definition.
- 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., Koster, 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: Standards Track
Expires: January 7, 2016

K. Li
Alibaba Group
A. Rahman
InterDigital
C. Bormann, Ed.
Universitaet Bremen TZI
July 06, 2015

Representing CoRE Formats in JSON and CBOR
draft-ietf-core-links-json-03

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.

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 7, 2016.

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:

- Canonical mapping
  - lossless round-tripping with [RFC6690] and between JSON and CBOR
  - but not trying for bit-preserving (DER-style) round-tripping

- The simplest thing that could possibly work
  - Do not cater for RFC 5988 complications caused by HTTP header character set issues [RFC2047]
- 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:

- the outer collection to an array of links;
- 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

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
         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)
         646573637269265646279    # "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       # "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 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, March 1997.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

7.2. Informative References

[I-D.greevenbosch-appsawg-cbor-cddl] Vigano, C. and H. Birkholz, "CBOR data definition language: a notational convention to express CBOR data structures.", draft-greevenbosch-appsawg-cbor-cddl-06 (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-03 (work in progress), June 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, November 1996.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[RFC7390] Rahman, A. and E. Dijk, "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, October 2014.

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
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2016

Z. Shelby
M. Koster
ARM
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant

October 16, 2015

CoRE Resource Directory
draft-ietf-core-resource-directory-05

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 April 18, 2016.

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 16
5.5. Read Endpoint Links 17
5.6. Update Endpoint Links 19
6. Group Function Set 21
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' 29
8.2. Export attribute 'exp' 30
9. DNS-SD Mapping 30
9.1. DNS-based Service discovery 30
9.2. mapping ins to <Instance> 31
9.3. Mapping rt to <ServiceType> 32
9.4. Domain mapping 32
9.5. TXT Record key=value strings 32
9.6. Importing resource links into DNS-SD 33
10. Security Considerations 34
10.1. Endpoint Identification and Authentication 34
10.2. Access Control 34
10.3. Denial of Service Attacks 34
11. IANA Considerations 35
11.1. Resource Types 35

11.2. Link Extension 35
11.3. RD Parameter Registry 35
12. Examples 36
12.1. Lighting Installation 36
12.1.1. Installation Characteristics 36
12.1.2. RD entries 38
12.1.3. DNS entries 41
12.1.4. RD Operation 44
12.2. OMA Lightweight M2M (LWM2M) Example 44
12.2.1. The LWM2M Object Model 45
12.2.2. LWM2M Register Endpoint 46
12.2.3. Alternate Base URI 47
12.2.4. LWM2M Update Endpoint Registration 48
12.2.5. LWM2M De-Register Endpoint 48
13. Acknowledgments 48
14. Changelog 48
15. References 51
15.1. Normative References 51
15.2. Informative References 52
Authors' Addresses 53

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]. This specification however 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.

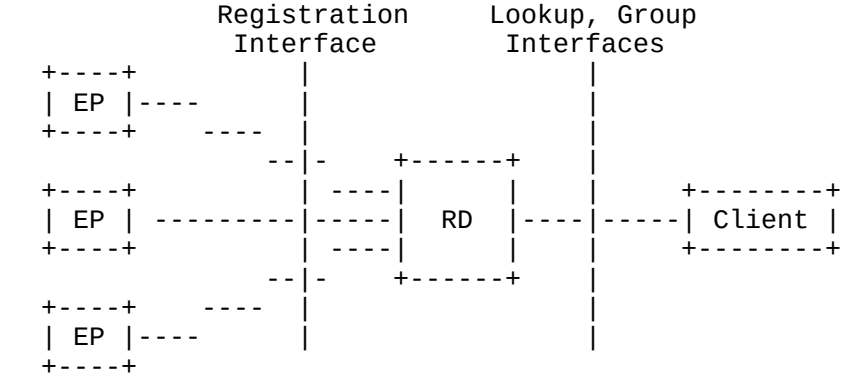


Figure 1: The resource directory architecture.

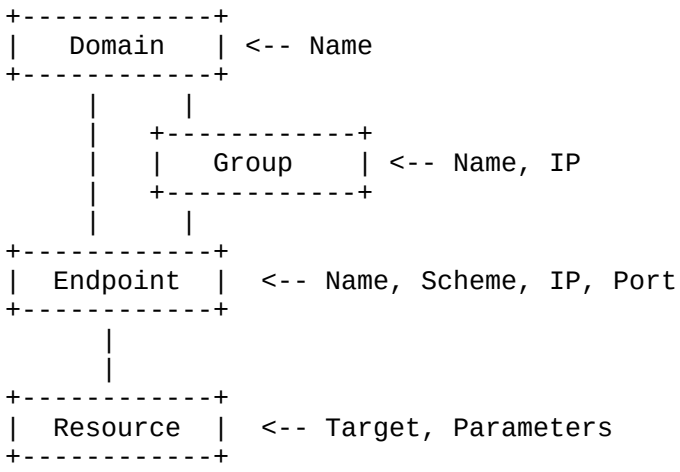


Figure 2: The resource directory information hierarchy.

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 link-format, link-format+cbor, or link-format+json 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 link-format, link-format+json, or link-format+cbor for storage and access by Resource Directories. Since it is common practice for these to be URN encoded, simple and lossless structural transforms will 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 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

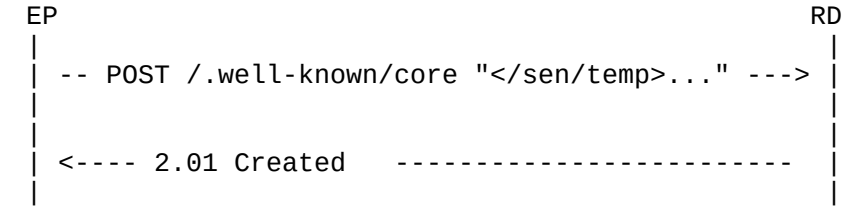
- empty, in which case the directory server is encouraged by this POST request to perform GET requests at the requesting server’s default discovery URI.

or

- 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.



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:

- specific static configuration (e.g., anycast addresses), if any,
- the ABRO option of 6LoWPAN-ND [RFC6775],
- other ND options that happen to point to servers (such as RDNSS),
- 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.

HTTP does not support multicast and consequently discovery has no HTTP interface.

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 the value "core.rd", "core.rd-lookup", "core.rd-group" or "core.rd*"

Content-Type: application/link-format (if any)

Content-Type: application/link-format+json (if any)

Content-Type: 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.

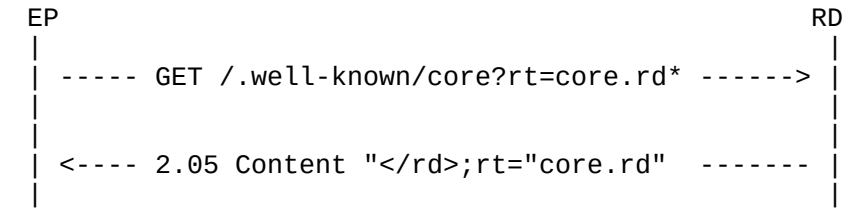
Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

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.



Req: GET coap://[ff02::1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd",
</rd-lookup>;rt="core.rd-lookup",
</rd-group>;rt="core.rd-group"

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-Type: application/link-format

Content-Type: application/link-format+json

Content-Type: application/link-format+cbor

The following response codes are defined for this interface:

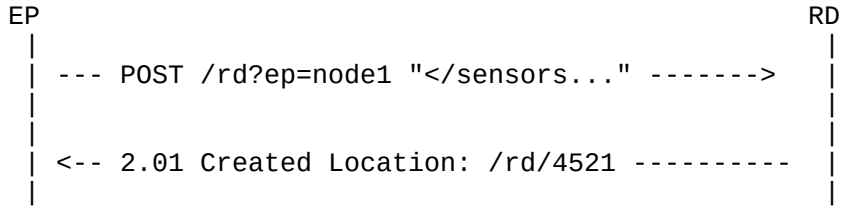
Success: 2.01 "Created" or 201 "Created". The Location header MUST be included with the new resource entry for the endpoint. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration. The resource returned in the Location is only for 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.




```
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-Type: 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-Type: application/link-format (optional)

Content-Type: application/link-format+json (optional)

Content-Type: application/link-format+cbor (optional)

The following response codes are defined for this interface:

Success: 2.04 "Changed" or 204 "No Content" in the update was successfully processed.

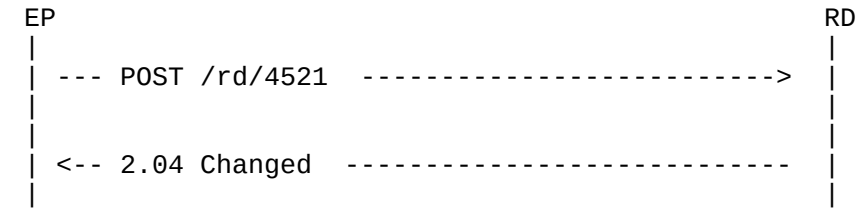
Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration 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

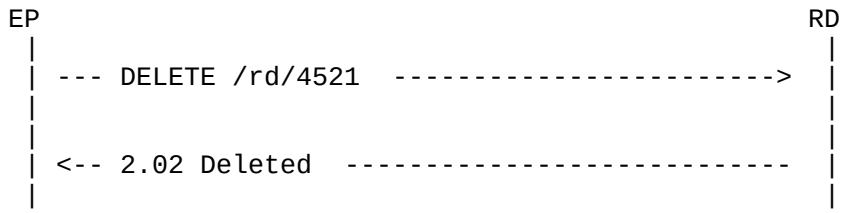
Failure: 4.00 "Bad Request" or 400 "Bad request". Malformed request.

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.

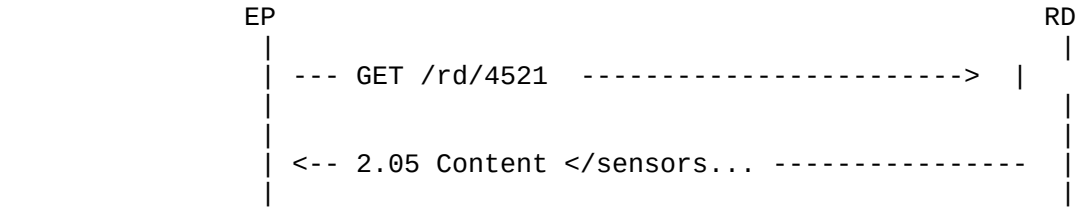
Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

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

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" 0r 204 "No Content" in the update was successfully processed.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

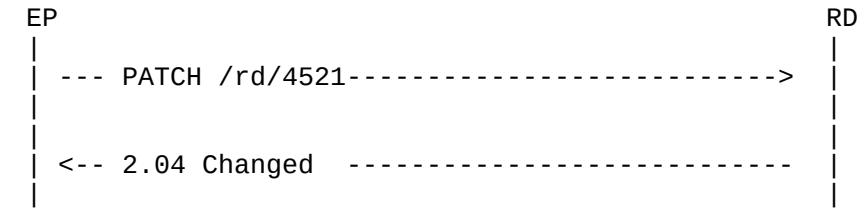
Failure: 4.04 "Not Found" or 404 "Not Found". Registration resource does not exist (e.g. may have expired).

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.



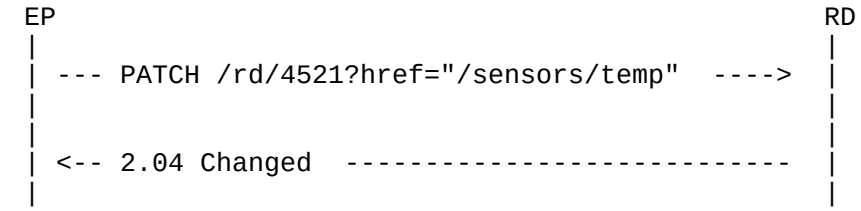
Req: PATCH /rd/4521

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.



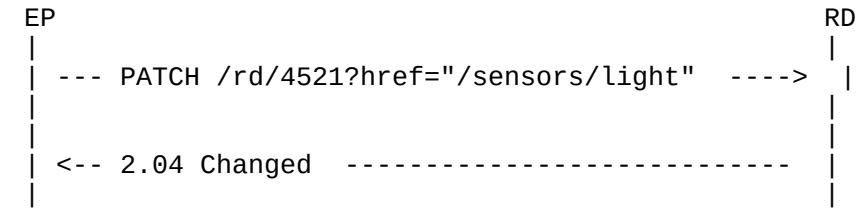
Req: PATCH /rd/4521?href="/sensors/temp"

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-Type: application/link-format

Content-Type: application/link-format+json

Content-Type: application/link-format+cbor

The following response codes are defined for this interface:

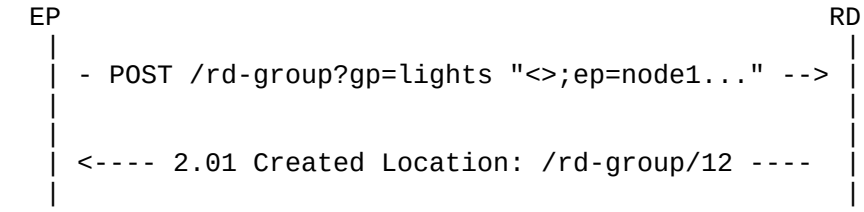
Success: 2.01 "Created" or 201 "Created". The Location header MUST be included with the new group entry. This Location MUST be a stable identifier generated by the RD as it is used for delete operations on this registration.

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.



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

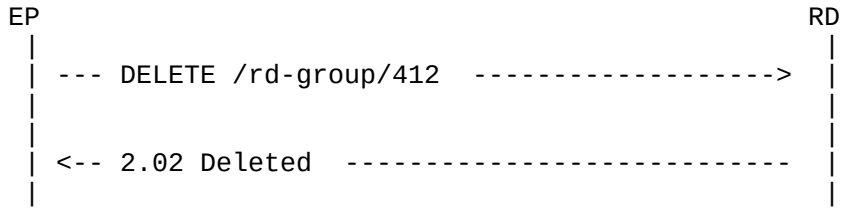
Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

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.

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.

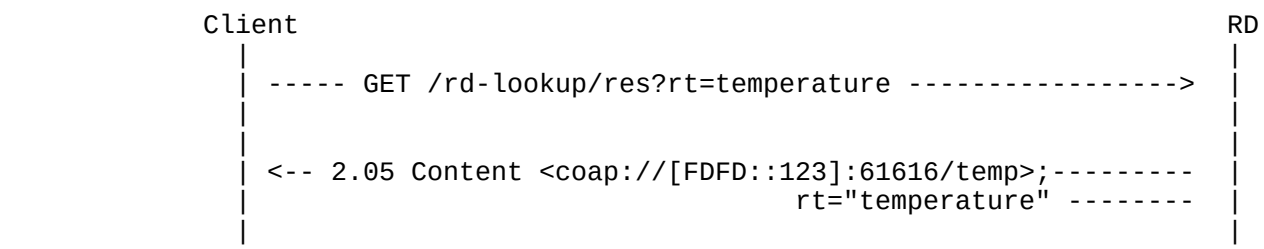
Failure: No error response to a multicast request.

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

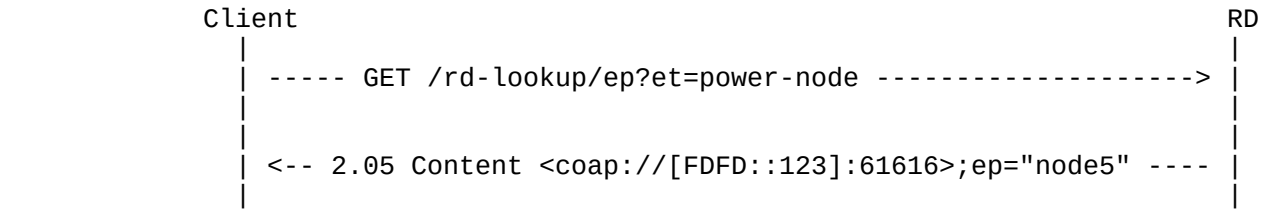
The examples in this section assume a CoAP host with IP address FDFD::123 and a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples. The following example shows a client performing a resource lookup:



Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content
<coap://[FDFD::123]:61616/temp>;rt="temperature"

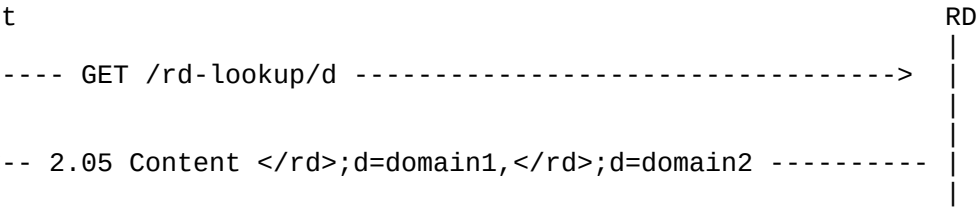
The following example shows a client performing an endpoint type lookup:



Req: GET /rd-lookup/ep?et=power-node

Res: 2.05 Content
<coap://[FDFD::123]:61616>;ep="node5",
<coap://[FDFD::123]:61616>;ep="node7"

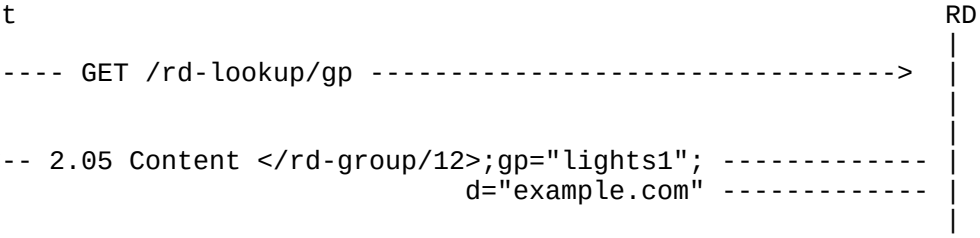
The following example shows a client performing a domain lookup:



Req: GET /rd-lookup/d

Res: 2.05 Content
</rd>;d="domain1",
</rd>;d="domain2"

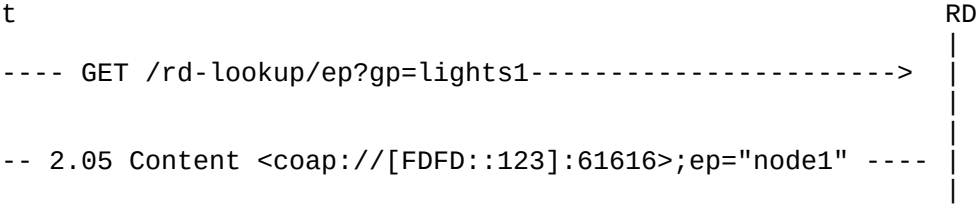
The following example shows a client performing a group lookup for all groups:



Req: GET /rd-lookup/gp

Res: 2.05 Content
</rd-group/12>;gp="lights1";d="example.com"

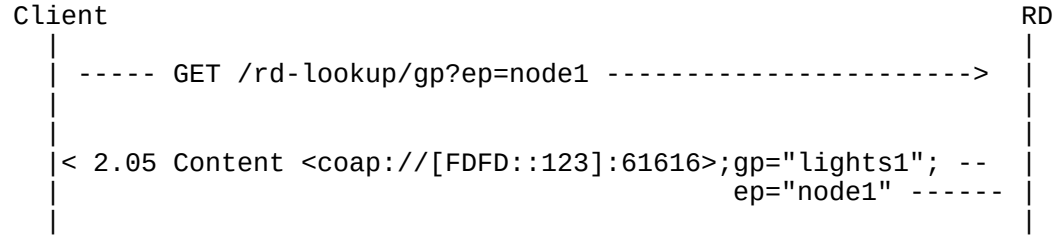
The following example shows a client performing a lookup for all endpoints in a particular group:



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:



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 means.

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.

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 "lamp._sub._dali._udp".

The 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).

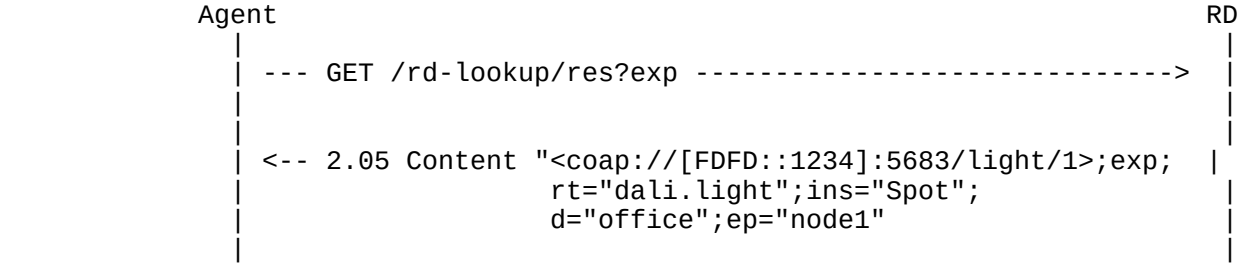
The resource <URI> is exported as key/value pair "path=<URI>".

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:




```
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		Name of the endpoint
Lifetime	lt	60-4294967295	Lifetime of the registration in seconds
Domain	d		Domain to which this endpoint belongs
Endpoint Type	et		Semantic name of the endpoint
Context	con	URI	The scheme, address and port at which this server is available
Endpoint Name	ep		Name of the endpoint, max 63 bytes
Group Name	gp		Name of a group in the RD
Page	page	Integer	Used for pagination
Count	count	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-type:application/link-format)<light/middle>,<light/left>

Res: 2.04 Changed

Req: POST
 coap://[FDFD::ABCD:2]/light/grp1
(content-type: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 it’s 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. 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 -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.

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-03 (work in progress), July 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, "CoRE Interfaces", draft-ietf-core-interfaces-03 (work in progress), July 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
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-576-1500 x11516
Email: Michael.Koster@arm.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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

C. Jennings
Cisco
Z. Shelby
ARM
J. Arkko
A. Keranen
Ericsson

October 19, 2015

Media Types for Sensor Markup Language (SENML)
draft-jennings-core-senml-02

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.

Table of Contents

1. Overview 3
2. Requirements and Design Goals 3
3. Terminology 4
4. Semantics 4
5. Associating Meta-data 6
6. JSON Representation (application/senml+json) 7
6.1. Examples 8
6.1.1. Single Datapoint 8
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) 11
8. XML Representation (application/senml+xml) 12
9. EXI Representation (application/senml-exi) 13
10. Usage Considerations 15
11. IANA Considerations 16
11.1. Units Registry 17
11.2. Media Type Registration 19
11.2.1. senml+json Media Type Registration 19
11.2.2. senml+cbor Media Type Registration 21
11.2.3. senml+xml Media Type Registration 22
11.2.4. senml-exi Media Type Registration 22
11.3. XML Namespace Registration 23
11.4. CoAP Content-Format Registration 23
12. Security Considerations 24
13. Privacy Considerations 24
14. Acknowledgement 24
15. References 24
15.1. Normative References 24
15.2. Informative References 26
Authors' Addresses 27

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. There are many types of more complex measurements and measurements that this media type would not be suitable for. A decision was made not to carry most of the meta data about the sensor in this media type to help reduce the size of the data and improve efficiency in decoding. Instead meta-data about a sensor resource can be described out-of-band using the CoRE Link Format [RFC6690]. 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").

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 base value object(s) and array(s) of entries. Each entry is an object that has attributes such as a unique identifier for the sensor, the time the measurement was made, and the current value. 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":23.5, "u":"Cel" }]]
```

In the example above, the first element of the root array is empty object since there are no base values. The second array inside the root array has a single measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a temperature 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.

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].

4. Semantics

Each SenML representation carries a single array that represents a set of measurements and/or parameters. This array contains a base object with several optional attributes described below and a mandatory array of one or more entries.

Base Name: This is a string that is prepended to the names found in the entries. This attribute is optional.

Base Time: A base time that is added to the time found in an entry. This attribute is optional.

Base Units: A base unit that is assumed for all entries, unless otherwise indicated. This attribute is optional.

Version: Version number of media type format. This attribute is optional positive integer and defaults to 2 if not present.

The measurement or parameter entries array contains values for sensor measurements or other generic parameters, such as configuration parameters. There must be at least one entry in the array. This array is called simply "measurement array" in the following text.

Each array entry contains several attributes, some of which are optional and some of which are mandatory:

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.

Units: Units for a measurement value. Optional.

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 ("bv" for "Boolean Value") and Strings ("sv" for "String Value"). Exactly one of these three fields MUST appear.

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: A 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. Optional.

The SenML format can be extended with further custom attributes placed in a base object, or in an entry. Extensions in a base object pertain to all entries following the base object, whereas extensions in an entry object only pertain to that entry.

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 it 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 a EUI-48 or EUI-64 identifier (a MAC address) or some other 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.

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. [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.

A measurement array MAY be followed by another base object and measurement array. The new base object can add, change, and/or remove base values from the previous base object(s). The new base values are applied to the following measurement arrays. Every base object MUST be followed by a measurement array, and hence base objects are found in the root array at even indexes and measurement arrays at odd indexes.

Representing the statistical characteristics of measurements 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 more 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 representations, this meta-data can be made available using the CoRE Link Format [RFC6690].

The CoRE Link Format provides a simple way to describe Web Links, and in particular allows a web server to describe resources it is

hosting. The list of links that a web server has available, can be discovered by retrieving the /.well-known/core resource, which returns the list of links in the CoRE Link Format. Each link may contain attributes, for example title, resource type, interface description, and content-type.

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.

Further semantics about a resource can be included in the Resource Type and Interface Description attributes. The Resource Type (rt=) attribute is meant to give a semantic meaning to that resource. For example rt="outdoor-temperature" would indicate static semantic meaning in addition to the unit information included in SenML. The Interface Description (if=) attribute is used to describe the REST interface of a resource, and may include e.g. a reference to a WADL description [WADL].

6. JSON Representation (application/senml+json)

Base object variables:

SenML	JSON	Type
Base Name	bn	String
Base Time	bt	Number
Base Units	bu	Number
Version	ver	Number

Measurement or Parameter Entries:

SenML	JSON	Notes
Name	n	String
Units	u	String
Value	v	Floating point
String Value	sv	String
Boolean Value	bv	Boolean
Value Sum	s	Floating point
Time	t	Number
Update Time	ut	Number

All of the data is UTF-8, but since this is for machine to machine communications on constrained systems, only characters with code points between U+0001 and U+007F are allowed which corresponds to the ASCII [RFC0020] subset of UTF-8.

The root content consists of an array with even amount of JSON objects where the first (and then every odd) element is a base object and the second (and then every even) element is a measurements array. The base object MAY contain a "bn" attribute with a value of type string. The object MAY contain a "bt" attribute with a value of type number. The object MAY contain a "bu" attribute with a value of type string. The object MAY contain a "ver" attribute with a value of type number. The object MAY contain other attribute value pairs. The base object MUST be followed by an array. The array MUST have one or more measurement or parameter objects.

If the root array has more than one base object, each following base object modifies the base values using the JSON merge patch format [RFC7396]. That is, base values can be added or modified by defining their new values and existing base values can removed by defining the value as "null".

Inside each measurement or parameter object the "n", "u", and "sv" attributes are of type string, the "t" and "ut" attributes are of type number, the "bv" attribute is of type boolean, and the "v" and "s" attributes are of type floating point. All the attributes are optional, but as specified in Section 4, one of the "v", "sv", or "bv" attributes MUST appear unless the "s" attribute is also present. The "v", and "sv", and "bv" attributes MUST NOT appear together.

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

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":23.5 }] ]
```

6.1.2. Multiple Datapoints

The following example shows voltage and current now, i.e., at an unspecified time. The device has an EUI-64 MAC address of 0024befffe804ff1.

```
[{"bn": "urn:dev:mac:0024befffe804ff1/"},
 [ { "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:mac:0024befffe804ff1/",
  "bt": 1276020076,
  "bu": "A",
  "ver": 1},
 [ { "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. In this situation the SenML stream can be sent and received in a partial fashion, i.e., a measurement entry can be read as soon as it is received and only not when the entire SenML object is complete.

For instance, the following stream of measurements may be sent from the producer of a SenML object to the consumer of that SenML object, and each measurement object may be reported at the time it arrives:

```
[{"bn": "http://[2001:db8::1]",
  "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": "http://[2001:db8::1]",
  "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 } ]
]
```

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": "http://[2001:db8::2]/",
  "bt": 1320078429,
  "ver": 1},
 [ { "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:

- For compactness, the CBOR representation uses integers for the map keys defined in Table 1. This table is conclusive, i.e., there is no intention to define any additional integer map keys; any extensions will use string map keys.

- 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	ver	-1
Base Name	bn	-2
Base Time	bt	-3
Base Units	bu	-4
Name	n	0
Units	u	1
Value	v	2
String Value	sv	3
Boolean Value	bv	4
Value Sum	s	5
Time	t	6
Update Time	ut	7

Table 1: CBOR representation: integers for map keys

8. XML Representation (application/senml+xml)

A SenML object 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.

```
<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml"
       bn="urn:dev:mac:0024befffe804ff1/"
       bt="1276020076"
       ver="1" bu="A">
  <e n="voltage" u="V" v="120.1" />
  <e n="current" t="-5" v="1.2" />
  <e n="current" t="-4" v="1.30" />
  <e n="current" t="-3" v="0.14e1" />
  <e n="current" t="-2" v="1.5" />
  <e n="current" t="-1" v="1.6" />
  <e n="current" t="0" v="1.7" />
</senml>
```

The RelaxNG schema for the XML is:

```
default namespace = "urn:ietf:params:xml:ns:senml"
namespace rng = "http://relaxng.org/ns/structure/1.0"

e = element e {
  attribute n { xsd:string }?,
  attribute u { xsd:string }?,
  attribute v { xsd:float }?,
  attribute sv { xsd:string }?,
  attribute bv { xsd:boolean }?,
  attribute s { xsd:decimal }?,
  attribute t { xsd:int }?,
  attribute ut { xsd:int }?,
  p*
}

senml =
  element senml {
    attribute bn { xsd:string }?,
    attribute bt { xsd:int }?,
    attribute bu { xsd:string }?,
    attribute ver { xsd:int }?,
    e*
  }

start = senml
```

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.

The following XSD Schema is generated from the RelaxNG and used for strict schema guided EXI processing.

```
<?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="e">
    <xs:complexType>
      <xs:attribute name="n" type="xs:string"/>
      <xs:attribute name="u" type="xs:string"/>
      <xs:attribute name="v" type="xs:float"/>
      <xs:attribute name="sv" type="xs:string"/>
      <xs:attribute name="bv" type="xs:boolean"/>
      <xs:attribute name="s" type="xs:decimal"/>
      <xs:attribute name="t" type="xs:int"/>
      <xs:attribute name="ut" type="xs:int"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="senml">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="ns1:e"/>
      </xs:sequence>
      <xs:attribute name="bn" type="xs:string"/>
      <xs:attribute name="bt" type="xs:int"/>
      <xs:attribute name="bu" type="xs:string"/>
      <xs:attribute name="ver" type="xs:int"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The following shows a hexdump of the EXI produced from encoding the following XML example. Note that while this example is similar to the first example in Section 6.1.2 in JSON format.

```
<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml"
       bn="urn:dev:ow:10e2073a01080063" >
  <e n="voltage" t="0" v="120.1" u="V" />
  <e n="current" t="0" v="1.2" u="A" />
</senml>
```

Which compresses to the following displayed in hexdump:

```
00000000  a0 30 0d 85 01 d7 57 26  e3 a6 46 57 63 a6 f7 73
00000010  a3 13 06 53 23 03 73 36  13 03 13 03 83 03 03 63
00000020  36 21 2e cd ed 8e 8c 2c  ec a8 00 00 d5 95 88 4c
00000030  02 08 4b 1b ab 93 93 2b  73 a2 00 00 34 14 19 00
00000040  c0
```

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:

```
<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml"
       bn="urn:dev:ow:10e2073a01080063" >
  <e n="temp" v="23.1" u="Cel" />
</senml>
```

The compressed form, using the byte alignment option of EXI, for the above XML is the following:

```
00000000  a054c9006c200000 1d75726e3a646576 |.T..l ...urn:dev|
00000010  3a6f773a31306532 3037336130313038 |:ow:10e2073a0108|
00000020  3030363304010674 656d70030543656c |0063...temp..Cel|
00000030  0100e70101000102                  |........|
```

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 0x14 and going to byte 0x23 with it’s device ID, and replacing the value "0xe7 0x01" at location 0x32 to 0x33 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 location 0x32 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 at location 0x33 is set to the integer temperature in tenths of degrees right shifted 7 bits. In this example 231 >> 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. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification.

11.1. Units Registry

IANA will create a registry of 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 [NIST811] and [BIPM].

All the registry entries in Table 2 use "v" (numeric) values. New entries allocated in the registry must define what kind of values they use.

Symbol	Description	Reference
m	meter	RFC-AAAA
kg	kilogram	RFC-AAAA
s	second	RFC-AAAA
A	ampere	RFC-AAAA
K	kelvin	RFC-AAAA
cd	candela	RFC-AAAA
mol	mole	RFC-AAAA
Hz	hertz	RFC-AAAA
rad	radian	RFC-AAAA
sr	steradian	RFC-AAAA
N	newton	RFC-AAAA
Pa	pascal	RFC-AAAA
J	joule	RFC-AAAA
W	watt	RFC-AAAA
C	coulomb	RFC-AAAA
V	volt	RFC-AAAA
F	farad	RFC-AAAA
Ohm	ohm	RFC-AAAA
S	siemens	RFC-AAAA
Wb	weber	RFC-AAAA
T	tesla	RFC-AAAA
H	henry	RFC-AAAA
Cel	degrees Celsius	RFC-AAAA
lm	lumen	RFC-AAAA
lx	lux	RFC-AAAA
Bq	becquerel	RFC-AAAA
Gy	gray	RFC-AAAA
Sv	sievert	RFC-AAAA
kat	katal	RFC-AAAA
pH	pH acidity	RFC-AAAA
%	Value of a switch (note 1)	RFC-AAAA
count	counter value	RFC-AAAA
%RH	Relative Humidity	RFC-AAAA
m2	area	RFC-AAAA

l	volume in liters	RFC-AAAA
m/s	velocity	RFC-AAAA
m/s2	acceleration	RFC-AAAA
l/s	flow rate in liters per second	RFC-AAAA
W/m2	irradiance	RFC-AAAA
cd/m2	luminance	RFC-AAAA
Bspl	bel sound pressure level	RFC-AAAA
bit/s	bits per second	RFC-AAAA
lat	degrees latitude (note 2)	RFC-AAAA
lon	degrees longitude (note 2)	RFC-AAAA
%EL	remaining battery energy level in percents	RFC-AAAA
EL	remaining battery energy level in seconds	RFC-AAAA
beat/m	Heart rate in beats per minute	RFC-AAAA
beats	Cumulative number of heart beats	RFC-AAAA

Table 2

- Note 1: A value of 0.0 indicates the switch is off while 100.0 indicates on.

- 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.

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].

11.2. Media Type Registration

The following registrations are done following the procedure specified in [RFC6838] and [RFC7303].

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification.

11.2.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC7159]. Specifically, only the ASCII [RFC0020] subset of the UTF-8 characters are allowed. 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

11.2.2. 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

11.2.3. senml+xml Media Type Registration

Type name: application

Subtype name: senml+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

11.2.4. 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

11.3. 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

11.4. 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 3.

Media type	ID
application/senml+json	TBD
application/senml+cbor	TBD
application/senml+xml	TBD
application/senml-exi	TBD

Table 3: CoAP Content-Format IDs

12. Security Considerations

See Section 13. Further discussion of security properties can be found in Section 11.2.

13. 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.

14. 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 was contributed by Carsten Bormann.

15. References

15.1. 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>.

[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>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <http://www.rfc-editor.org/info/rfc7396>.

[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>.

15.2. 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.

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <http://www.rfc-editor.org/info/rfc20>.

[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>.

[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>.

[WADL] Hadley, M., "Web Application Description Language (WADL)", 2009, <http://java.net/projects/wadl/sources/svn/content/trunk/www/wadl20090202.pdf>.

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: April 21, 2016

M. Koster
ARM Limited
A. Keranen
J. Jimenez
Ericsson

October 19, 2015

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-koster-core-coap-pubsub-03

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.

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 6
4.3. PUBLISH 8
4.4. SUBSCRIBE 9
4.5. UNSUBSCRIBE 11
4.6. READ 12
4.7. REMOVE 14
5. CoAP pub/sub Operation with Resource Directory 15
6. Sleep-Wake Operation 15
7. Simple Flow Control 16
8. Security Considerations 16
9. IANA Considerations 17
9.1. Resource Type value 'core.ps' 17
9.2. Response Code value '2.04' 17
9.3. Response Code value '4.29' 18
10. Acknowledgements 18
11. References 18
11.1. Normative References 18
11.2. Informative References 19
Authors' Addresses 19

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.

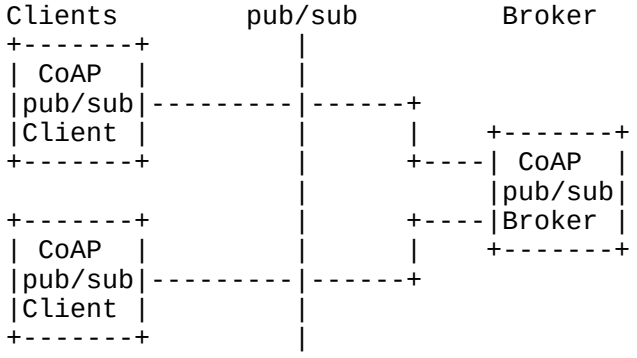


Figure 1: CoAP pub/sub Architecture

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.

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. With the exception of PUBLISH, all operations in the CoAP pub/sub Function Set MUST use confirmable (CON) CoAP messages.

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".

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: /.well-known/core{?rt}

URI Template Variables:

rt := Resource Type (optional). MAY contain the value "core.ps"

Content-Format: application/link-format (if any)

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 link subject variable whenever 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.

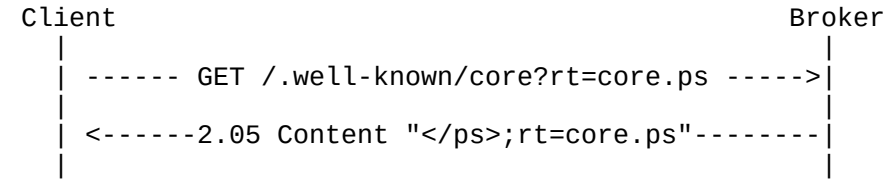


Figure 2: Example of DISCOVER

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 MUST use the CoRE link format [RFC6690]. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Type) relation 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.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

URI Template: /{+ps}

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 3 shows an example of a topic called "topic1" being successfully created.

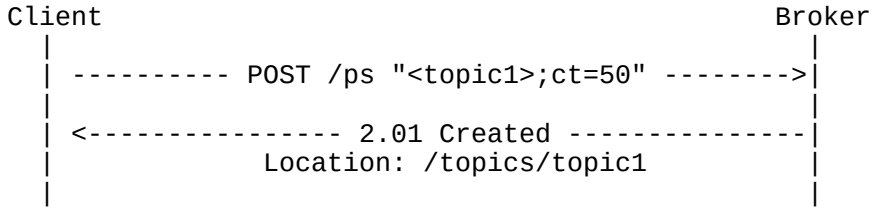


Figure 3: Example of CREATE

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. A client MAY use confirmable (CON) or non-confirmable (NON) messages to publish updates to a broker. 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 5. If a client publishes to a broker using non-confirmable messages, the broker MAY notify subscribed clients using non-confirmable messages. If a client publishes to a broker using confirmable messages, the broker MAY also notify all subscribed clients using confirmable messages. 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}

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 4 shows an example of a new value being successfully published to the topic "topic1". See Figure 5 for an example of a broker forwarding a message from a publishing client to a subscribed client.

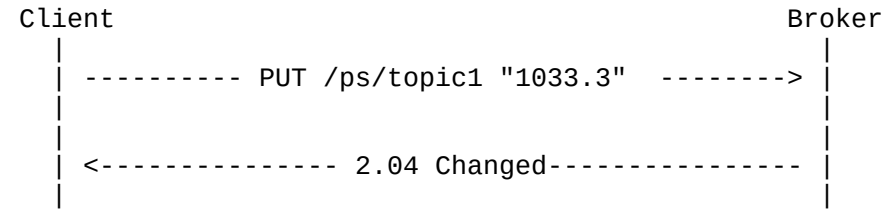


Figure 4: Example of PUBLISH

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.

The SUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

URI Template: /{+ps}/{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 5 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.

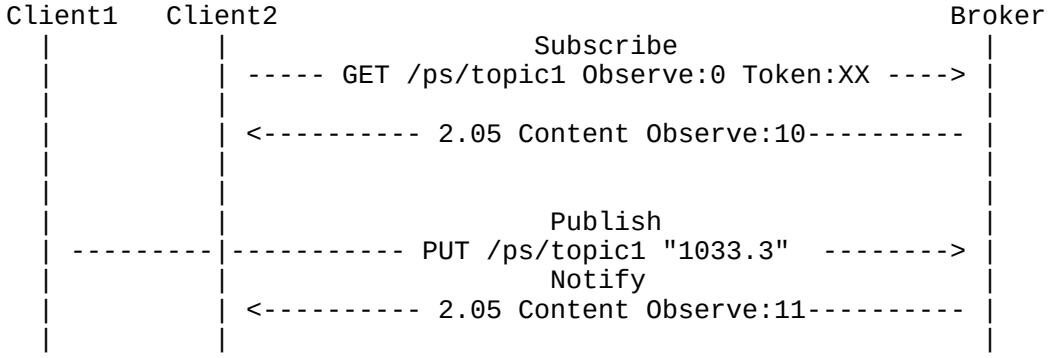


Figure 5: Example of SUBSCRIBE

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 [RFC7641].

The UNSUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

URI Template: /{+ps}/{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 6 shows an example of a client unsubscribe using the Observe=1 cancellation method.

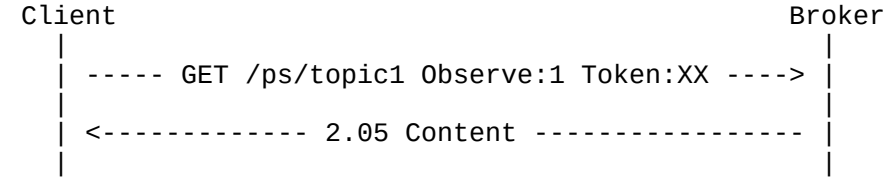


Figure 6: Example of UNSUBSCRIBE

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}

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 7 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.

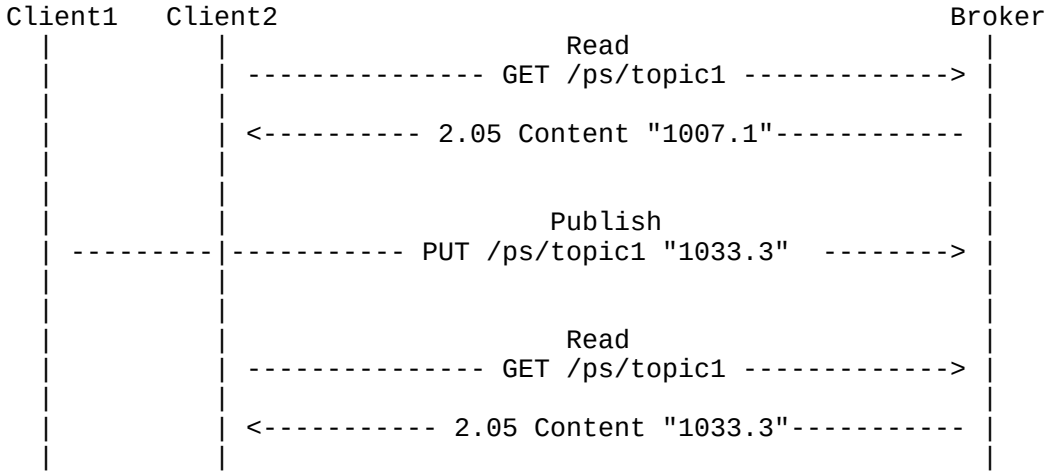


Figure 7: Example of READ

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.

The REMOVE interface is specified as follows:

Interaction: Client -> Broker

Method: DELETE

URI Template: /{+ps}/{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 8 shows a successful remove of topic1.

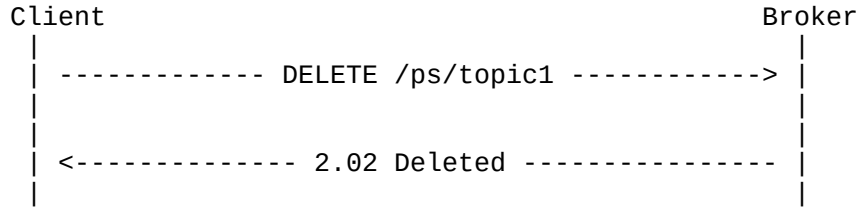


Figure 8: Example of REMOVE

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 confidentially 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 receival, 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’

- Attribute Value: core.ps
- Description: Section 4 of [[This document]]
- Reference: [[This document]]
- Notes: None

9.2. Response Code value ’2.04’

- Response Code: 2.04

- Description: Add No Content response to GET to the existing definition of the 2.04 response code.
- Reference: [[This document]]
- Notes: None

9.3. Response Code value '4.29'

- Response Code: 4.29
- Description: This error code is used by a server to indicate that a client is making too many requests on a resource.
- Reference: [[This document]]
- Notes: None

10. Acknowledgements

The authors would like to thank Hannes Tschofenig, Zach Shelby, Mohit Sethi, Peter van der Stok, Tim Kellogg, Anders Eriksson, and Goran Selander for their contributions and reviews.

11. References

11.1. 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.

[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: April 21, 2016

J. Mattsson
J. Fornehed
G. Selander
F. Palombini
Ericsson

October 19, 2015

Controlling Actuators with CoAP
draft-mattsson-core-coap-actuators-00

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 or OSCOAP is not enough. We describe several serious attacks any on-path attacker can do, and discuss 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.

Table of Contents

1. Introduction 2
2. Attacks 2
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. References 13
6.1. Normative References 14
6.2. 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). Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [RFC6347] or OSCOAP [I-D.selander-ace-object-security] to protect CoAP, but we show that this is not enough. 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 both DTLS 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 in specified.

2. Attacks

Internet-of-Things (IoT) deployments valuing security and privacy, MUST use a security protocol such as DTLS 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 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 or OSCOAP is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS, 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.

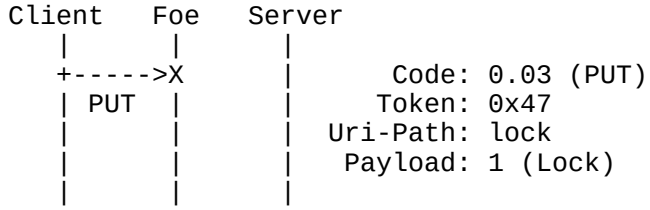


Figure 1: Blocking a Request

Where 'X' means the attacker is blocking delivery of the message.

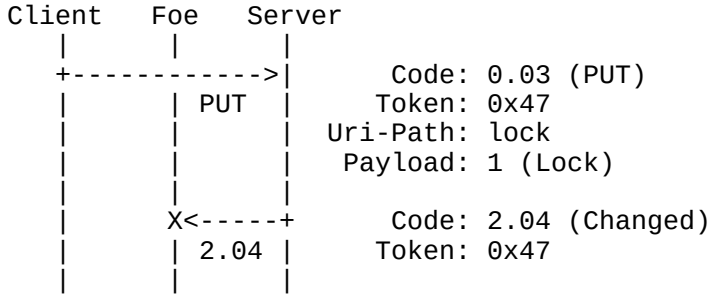


Figure 2: Blocking a Response

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 Schroedinger'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: In actuator deployments where confirmation is important, the application MUST notify the user upon reception of the response, or warn the user when a response is not received. The application SHOULD also indicate to the user that the status of the actuator is now uncertain.

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. This is true even if DTLS or OSCOAP is used, as long as the delayed packet is delivered inside the replay window. The replay window has a default length of 64 in DTLS and is application dependent in 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.

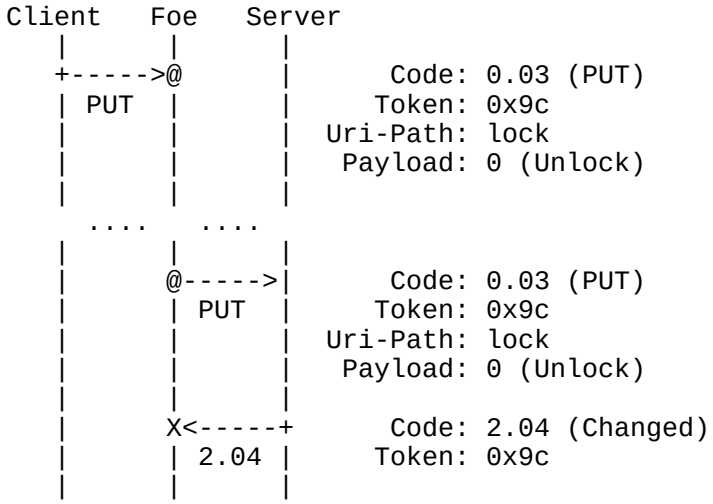


Figure 3: Delaying a Request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in spacetime).

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 in DTLS and unspecified in OSCOAP), 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.

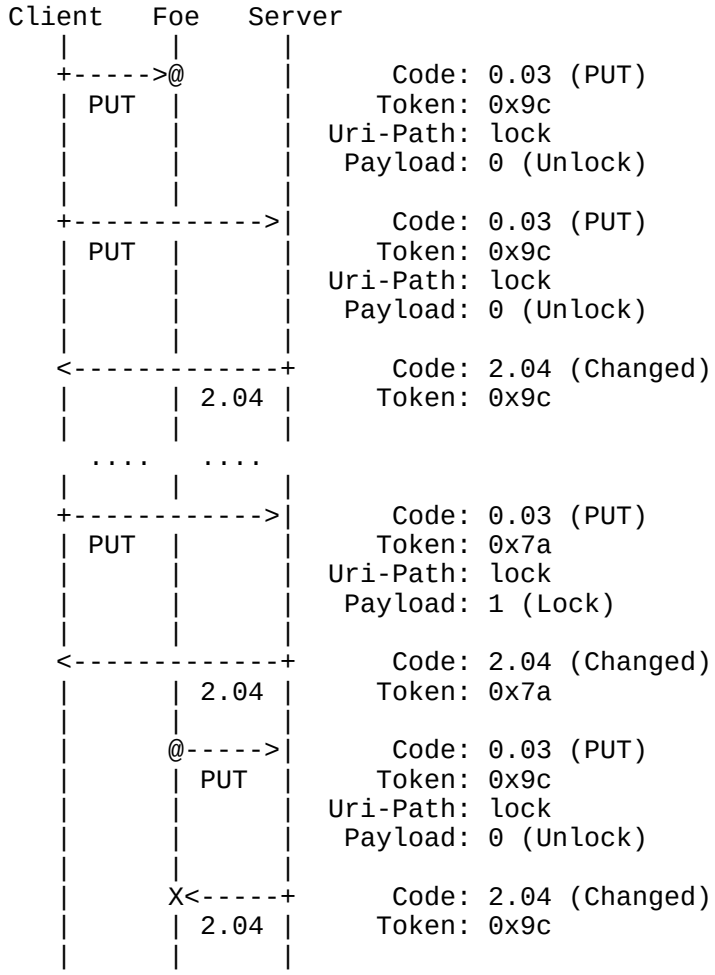


Figure 4: Delaying Request with Reordering

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 or by exchanging timestamps. 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. One such mechanism, the CoAP Repeat Option, is specified in Section 3.

Remedy: The CoAP Repeat 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] 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.

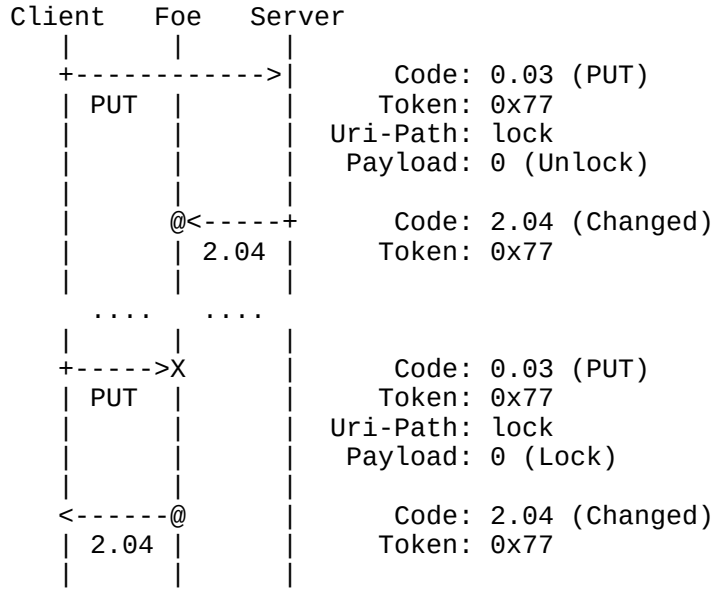


Figure 5: Delaying and Mismatching Response to PUT

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.

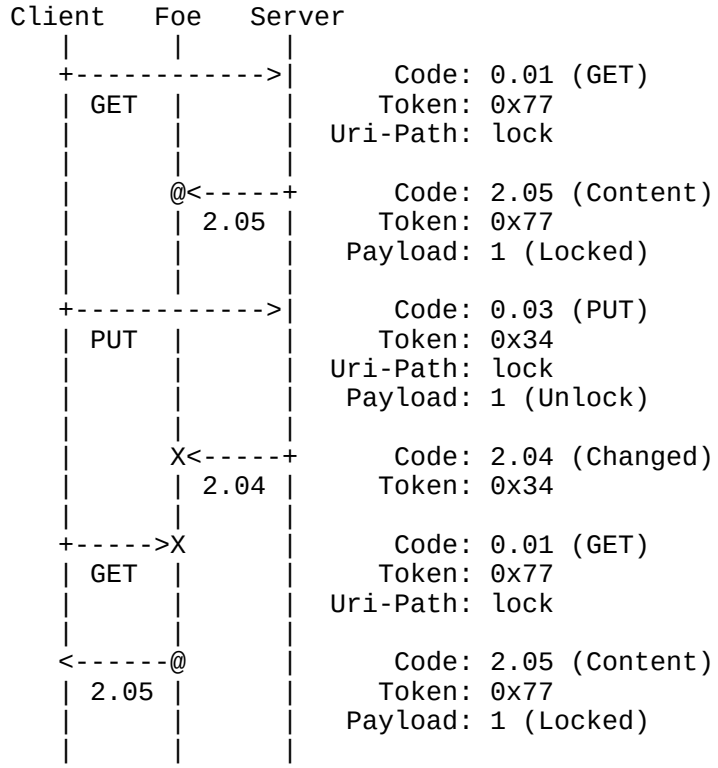


Figure 6: Delaying and Mismatching Response to GET

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).

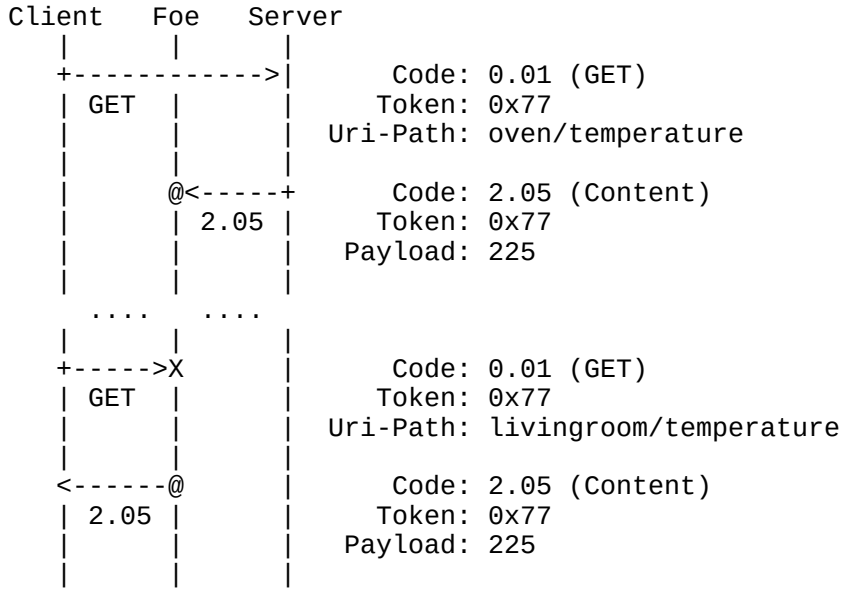


Figure 7: Delaying and Mismatching Response from other resource

OSCOAP is not susceptible to these attacks since it provides a secure binding between request and response messages.

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.

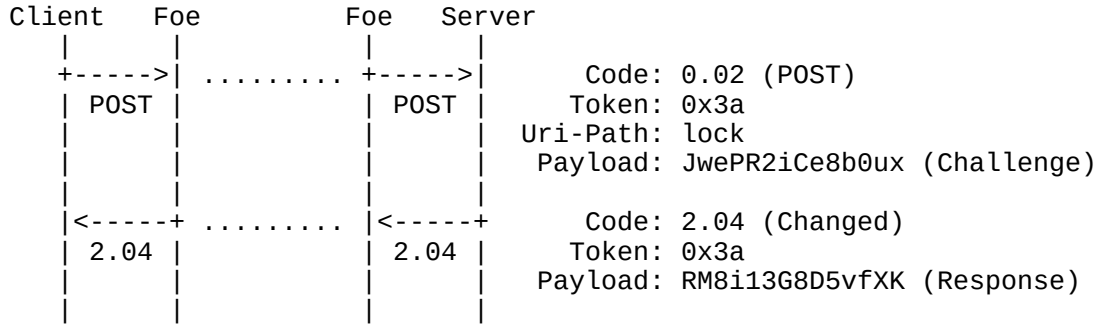


Figure 8: Relay Attack (the client is the actuator)

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.

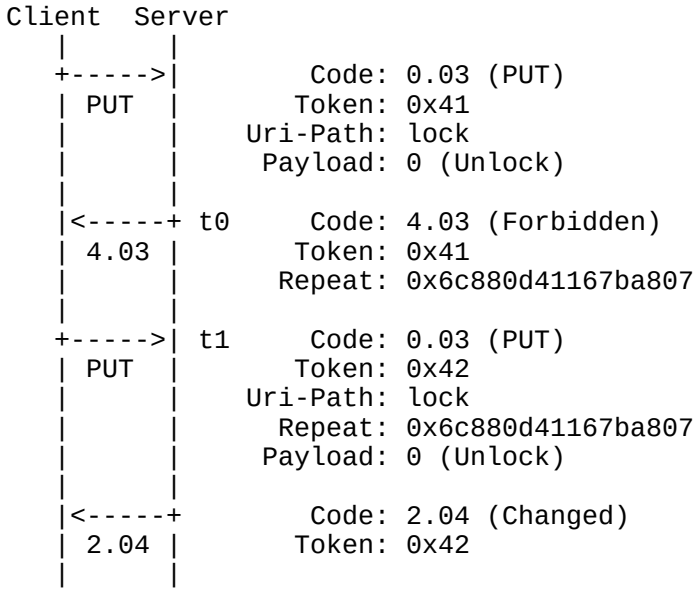


Figure 9: The Repeat Option

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 t0.

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 = (t1 - t0), where t1 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 sent. The threshold T is application specific.

An attacker able to control the server's clock with high precision, could still be able to perform a delay attack by moving the server's clock back in time, thus making the measured round-trip time smaller than the actual round-trip time. The times t0 and t1 MUST therefore be measured with a steady clock (one that cannot be adjusted).

EDITORS NOTE: The mechanism described above 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 t0. Other solutions involving time may be discussed:

- The server may simply send the client the current time in its timescale, i.e. a timestamp (option value = t0). 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.

- The server may instead of a pseudorandom value send an encrypted timestamp (option value = E(k, t0)). CTR-mode would from a security point be like sending (value = t0). 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, t0, parts of request)). This approach does not reduce the number of round-trips but eliminates server state.

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. 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>.

6.2. Informative References

[I-D.selander-ace-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "June 29, 2015", draft-selander-ace-object-security-02 (work in progress), June 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>.

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 3, 2016

T. Savolainen
Nokia
K. Hartke
Universitaet Bremen TZI
B. Silverajan
Tampere University of Technology
October 1, 2015

CoAP over WebSockets
draft-savolainen-core-coap-websockets-05

Abstract

This document specifies how to retrieve and update CoAP resources using CoAP requests and responses over the WebSocket Protocol.

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 11
6.1. Normative References 11
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.

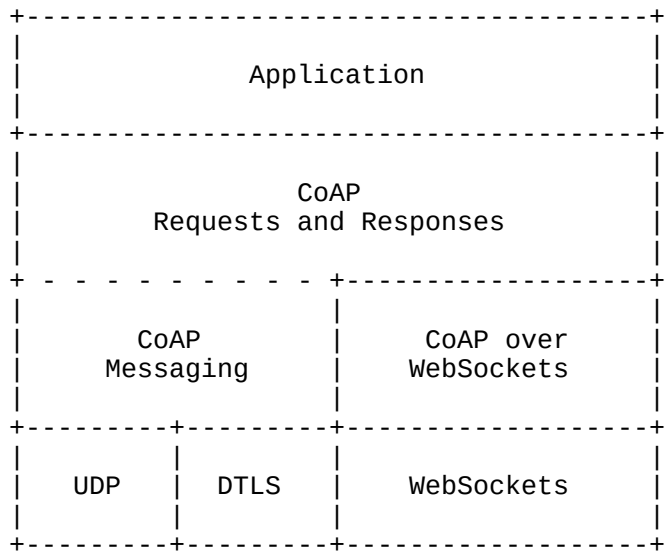


Figure 1: Abstract layering of CoAP extended by WebSockets

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.

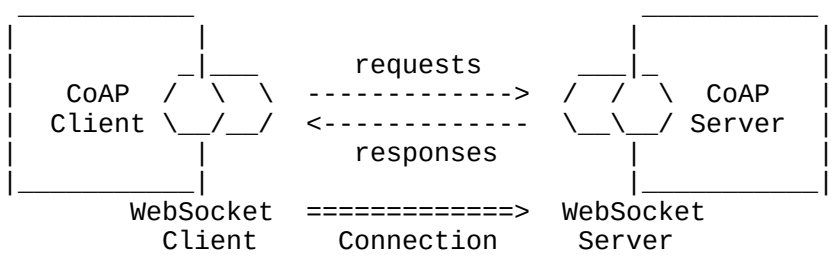


Figure 2: CoAP client (WebSocket client) accesses CoAP server (WebSocket server)

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:

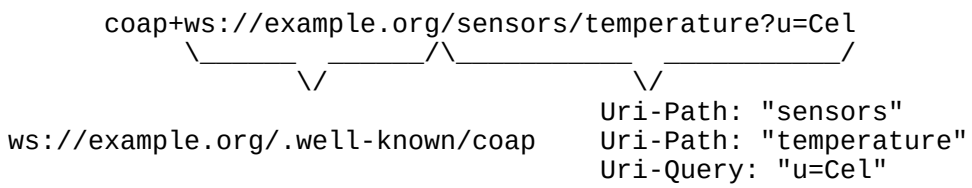


Figure 3: The "coap+ws" URI Scheme

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.

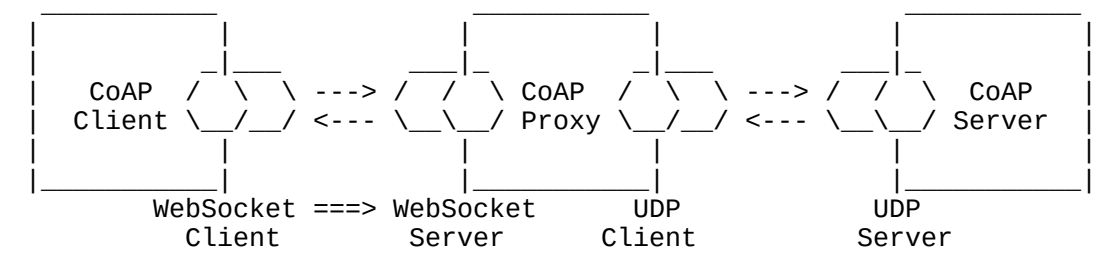


Figure 4: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

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.

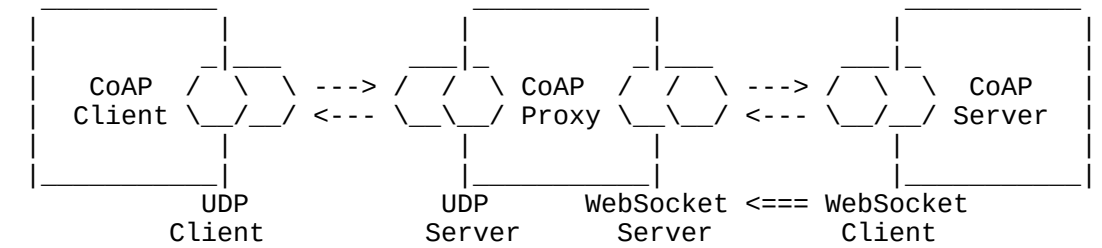


Figure 5: CoAP Client (UDP client) accesses sleepy CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

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:

- 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.
- Furthermore, since the CoAP version is already negotiated during the opening handshake, the "Ver" field can be elided as well.

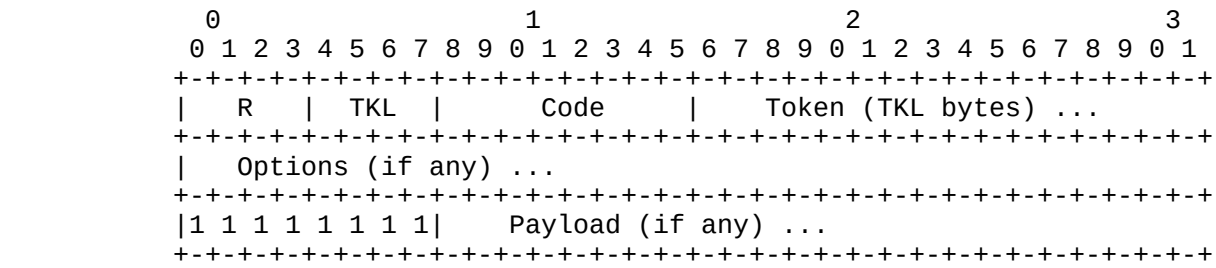


Figure 7: CoAP Message Format over WebSockets

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

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.

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.

Contact.
IETF Chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
This document.

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.

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.

Contact.
IETF Chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
This document.

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.
This document.

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).
This document.

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.

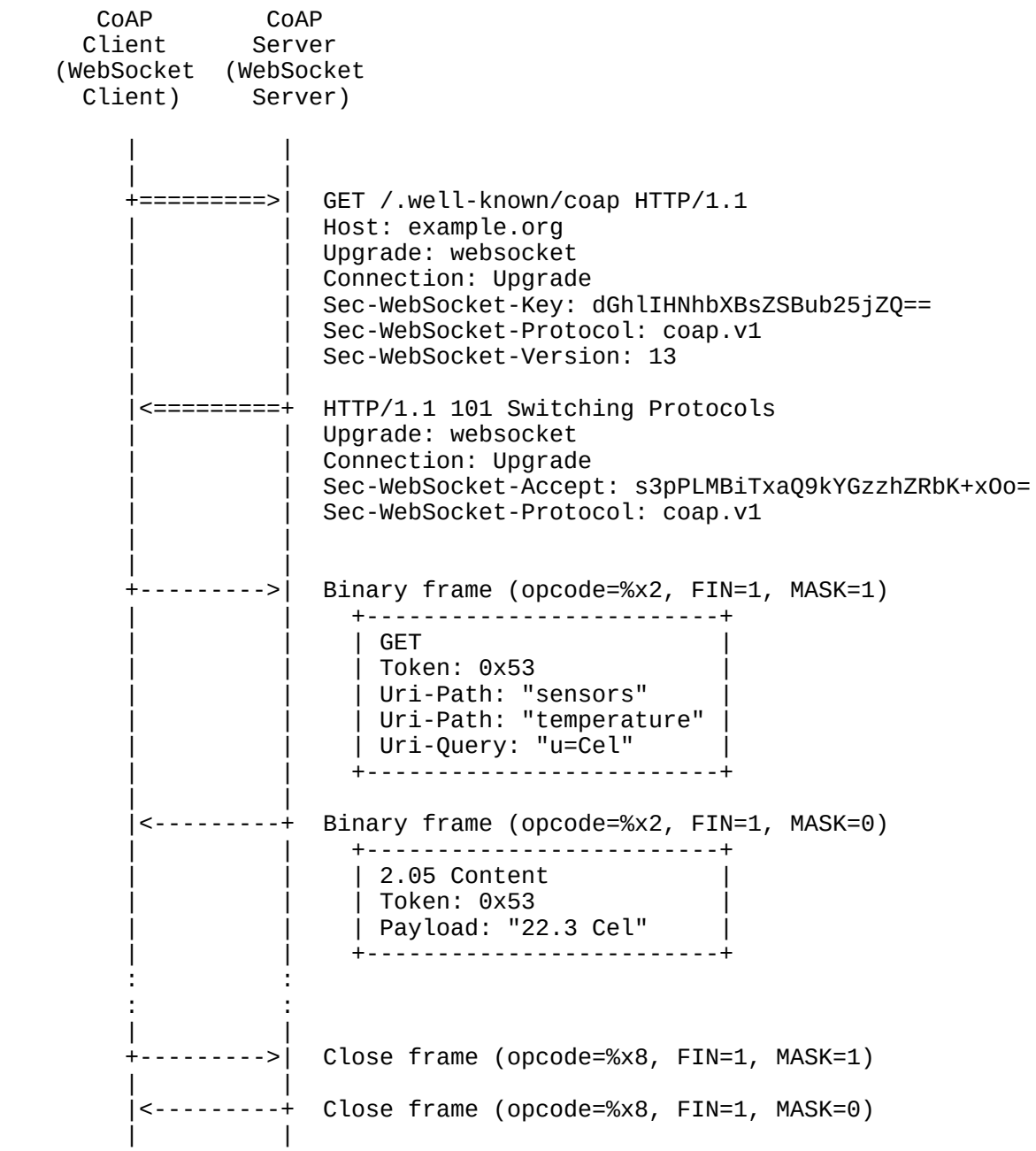


Figure 8: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

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.

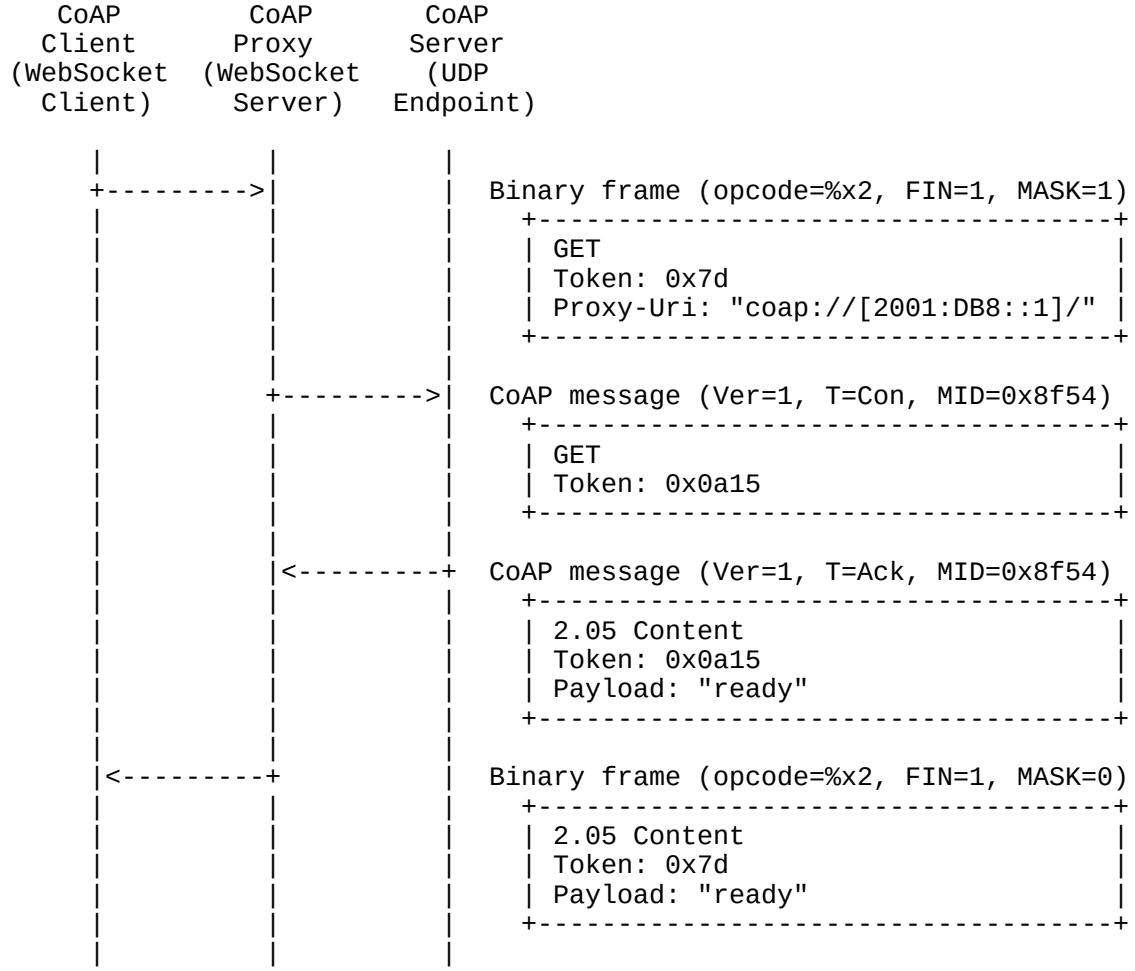


Figure 9: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

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

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

G. Selander
J. Mattsson
F. Palombini
Ericsson
L. Seitz
SICS Swedish ICT

October 19, 2015

Object Security of CoAP (OSCOAP)
draft-selander-ace-object-security-03

Abstract

This memo defines Object Security of CoAP (OSCOAP), a method for protection of request and response message exchanges of the Constrained Application Protocol (CoAP) using data object security. OSCOAP provides end-to-end encryption, integrity and replay protection to CoAP payload, options and header fields, and a secure binding between CoAP request and response messages. The use of OSCOAP is signaled with the Object-Security option, 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 21, 2016.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Terminology 4
2. Background 5
3. The Object-Security Option 6
4. Secure Message Format 7
4.1. Secure Message Header 7
4.2. Secure Message Body and Tag 8
4.2.1. Integrity Protection Only 8
4.2.2. Encryption and Integrity Protection 8
5. CoAP Message Protection 9
5.1. Integrity Protection Only 9
5.1.1. Protected CoAP message formatting 9
5.1.2. Secure Message formatting 10
5.1.3. Integrity Protection and Verification 10
5.2. Encryption and Integrity Protection 10
5.2.1. Protected CoAP message formatting 10
5.2.2. Secure Message formatting 11
6. Protected CoAP Message Fields 12
6.1. Protected CoAP Header Fields 12
6.2. Protected CoAP Options 12
6.2.1. Integrity Protection 13
6.2.2. Encryption 15
7. Replay Protection and Freshness 15
7.1. Replay Protection 15
7.2. Freshness 16
8. Security Considerations 16
9. Privacy Considerations 18
10. IANA Considerations 18
11. Acknowledgments 19
12. References 19
12.1. Normative References 19
12.2. Informative References 20
Appendix A. COSE Profile of SM 21
A.1. Integrity Protection Only 21
A.1.1. COSE_Sign 21
A.1.2. COSE_mac 22
A.2. Encryption and Integrity Protection: COSE_enveloped . . . 22
A.3. COSE Optimizations 23
Appendix B. Comparison of message sizes 25

B.1. MAC Only 26
B.2. Signature Only 27
B.3. Authenticated Encryption with Additional Data (AEAD) . . 29
B.4. Symmetric Encryption with Asymmetric Signature (SEAS) . . 30
Appendix C. Object Security of Content (OSCON) 32
C.1. Security Considerations of OSCON 33
Appendix D. Examples 34
D.1. CoAP Message Protection 34
D.1.1. Integrity Protection of CoAP Message Exchange 34
D.1.2. Additional Encryption of CoAP Message 36
D.2. Payload Protection 37
D.2.1. Proxy Caching 37
D.2.2. Publish-Subscribe 38
D.2.3. Transporting Authorization Information 40
Authors' Addresses 41

1. Introduction

The Constrained Application Protocol CoAP [RFC7252] was designed with a constrained RESTful environment in mind. CoAP references DTLS [RFC6347] for securing the message exchanges. Two prominent features of CoAP, store-and-forward and publish-subscribe exchanges, are problematic to secure with DTLS and transport layer security. As DTLS offers hop-by-hop security, in case of store-and-forward exchanges it necessitates a trusted intermediary. Securing publish-subscribe CoAP exchanges with DTLS requires the use of the keep-alive mechanism which incurs additional overhead and actually takes away most of the benefits of asynchronous communication.

The pervasive monitoring debate has illustrated the need to protect data also from trustworthy intermediary nodes as they can be compromised. The community has reacted strongly to the revelations, and new solutions must consider this attack [RFC7258] and include encryption by default.

This memo defines Object Security of CoAP (OSCOAP) a data object based communication security solution complementing DTLS and supporting secure messaging end-to-end across intermediary nodes. OSCOAP may be used in very constrained settings where DTLS cannot be supported. OSCOAP can also be combined with DTLS thus enabling, for example, 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.

OSCOAP provides end-to-end encryption, integrity and replay protection to CoAP payload, options and header fields, and a secure binding between CoAP request and response messages. Using this method the unprotected CoAP message is transformed into a protected

CoAP message, which contains a secure data object protecting the unprotected message, and which is sent instead of the unprotected message. The use of OSCOAP is signaled with the Object-Security option, also defined in this memo.

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.

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify". For "signature", see below.

RESTful terms, such as "resource" or "representation", are to be understood as used in HTTP [RFC7231] and CoAP.

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

Terminology for authentication and authorization in constrained environments, such as "Authorization Server", "Resource Server", etc, is defined in [I-D.ietf-ace-actors].

The CoAP option Object-Security and the Secure Message (SM) format are defined in this memo.

Two different scopes of object security are defined:

- OSCOAP = object security of CoAP, signaled with the Object-Security option
- OSCON = object security of content, signaled with Content Format/ Media Type set to application/oscon.

OSCON is defined in Appendix C and included for comparison with OSCOAP.

The COSE message format is defined in [I-D.ietf-cose-msg].

2. Background

The background for this work is provided by the use cases and architecture in [I-D.ietf-ace-usecases] and [I-D.ietf-ace-actors]. The focus of this memo is on end-to-end security in constrained environments in the presence of intermediary nodes.

For constrained-node networks there may be several reasons for messages to be cached or stored in one node and later forwarded.

For example, connectivity between the nodes may be intermittent, or some node may be sleeping at the time when the message should have been forwarded (see e.g. [I-D.ietf-ace-usecases] sections 2.1.1, and 2.5.1). Also, the architectural model or protocol applied may require an intermediary node which breaks security on transport layer (see e.g. [I-D.ietf-ace-usecases] sections 2.1.1, and 2.5.2). Examples of intermediary nodes include forward proxies, reverse proxies, pub-sub brokers, HTTP-CoAP cross-proxies, and SMS servers.

Based on these examples the following security requirements have been identified:

1. The payload shall be integrity protected and should be encrypted end-to-end from sender to receiver.

2. It shall be possible for an intended receiver to detect if it has received this message previously, i.e. replay protection.

3. The CoAP options which are not intended to be changed by an intermediary node shall be integrity protected between Client and Server.

4. The CoAP options which are not intended to be read by an intermediary node shall be encrypted between Client and Server.

5. The CoAP header fields "Code" and "Version" shall be integrity protected between Client and Server.

6. A Client shall be able to verify that a message is the response to a particular request the Client made.

In this list above, requirements 1-2 deals essentially with protecting the CoAP payload only, whereas 3-6 deals with protecting an entire CoAP request-response exchange, including also CoAP options and header fields.

Object Security of CoAP (OSCOAP), which is the main focus of this memo, addresses all requirements above by defining a method for

encryption, integrity protection and replay protection of CoAP payload, options and header fields, and a secure binding between CoAP request and response messages. OSCOAP consists of:

- the Object-Security option, indicating that OSCOAP is being used;
- a compact cryptographic message format called "Secure Message", based on the COSE message format ([I-D.ietf-cose-msg]); and
- a scheme for transforming an unprotected CoAP message into a protected CoAP message, which contains the Object-Security option and a Secure Message protecting CoAP payload, options and header fields.

The same method can be applied to payload only of individual messages, targeting only requirements 1-2 above. We call this object security of content (OSCON) and it is defined in Appendix C.

Examples of the use of OSCOAP and OSCON are given in Appendix D.

3. The Object-Security Option

In order to end-to-end protect CoAP message exchanges including options and headers, a new CoAP option is introduced: the Object-Security option. The Object-Security option indicates that OSCOAP is used, i.e. that certain CoAP Header fields, Options and Payload (if present) are integrity and replay protected and potentially encrypted, using a cryptographic message format called the Secure Message format Section 4.

This option is critical, safe to forward, it is not part of a cache key, and it is not repeatable. Figure 1 illustrates the structure of this option.

No.	C	U	N	R	Name	Format	Length
TBD	x		x		Object-Security	opaque	0, TBD

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Figure 1: The Object-Security Option

The length of the option depends on the specific choice of the Secure Message format. Length 0 indicates that the Secure Message is the CoAP Payload of the message, and is used when the CoAP message type used supports payload.

4. Secure Message Format

There exist already standardized and draft content formats for encryption and integrity protection of data such as CMS [RFC5652], JWS [RFC7515], JWE [RFC7516], and COSE [I-D.ietf-cose-msg].

Current CMS and JWx objects are undesirably large for very constrained devices. Large messages has a negative impact on memory and storage in constrained devices, packet fragmentation in constrained-node networks due to limited frame sizes, and increased energy consumption due to more data transmission and reception. The candidate for use with object security of CoAP messages is the COSE message format [I-D.ietf-cose-msg].

Pending an optimized and stable version of the COSE message format this memo defines the SM format to refer to a content format for encrypted and integrity protected data, and also includes a unique transaction identifier for replay protection. Appendix A shows a profile of the COSE message format which complies with the Secure Message format.

A Secure Message (SM) SHALL consist of Header, Body and Tag.

4.1. Secure Message Header

The following parameters SHALL be included in the SM Header:

- Context Identifier (CID). This parameter identifies the sender security context including the cipher suite, key(s) and additional algorithm specific parameters used to protect the message. Each client and server communicating using OSCOAP has two contexts, one for sending and one for receiving.

- Sequence Number (SEQ). The Sequence Number parameter enumerates the Secure Messages sent associated to a Context Identifier, and is used for replay protection and uniqueness of nonce. The start sequence number SHALL be 0. For a given key, any Sequence Number MUST NOT be used more than once.

The granularity of "sender" - what is being identified with the Context Identifier - is defined by the application. With OSCOAP the Context Identifier typically identifies the sending party and different resources may be identified by the Uri-Path in the request. (Compare Appendix C.)

The ordered sequence (SEQ, CID) is called Transaction Identifier (TID), and SHALL be unique for each SM.

4.2. Secure Message Body and Tag

The use cases require support for two message types, one for Encryption and Integrity Protection, and another for integrity protection only. The SM Body and the SM Tag are different depending on message type.

For Integrity Protection Only we denote by Authenticated Data (AD) the data which is integrity protected in the Secure Message. For Encryption and Integrity Protection we denote by Plaintext and Additional Authenticated Data (AAD), the data which is encrypted and integrity protected, and integrity protected only, respectively, in the Secure Message.

The message type SHALL be explicit to allow an intermediate node to distinguish between the two types and read the SM Body of an Integrity Protected Only message.

4.2.1. Integrity Protection Only

In the case of integrity protection only, the SM Body SHALL consist of the payload of the CoAP message.

The SM Tag SHALL consist of the Signature / Message Authentication Code (MAC) as defined by the cipher suite calculated over the Authenticated Data (AD). The AD for OSCOAP is defined in Section 5.1.2.

4.2.2. Encryption and Integrity Protection

The use cases require support for two kinds of cipher suites: Authenticated Encryption with Additional Data (AEAD) as well as Symmetric Encryption and Asymmetric Signature (SEAS).

In case of AEAD, the SM Body and SM Tag SHALL consist of the Ciphertext as defined by the cipher suite calculated over the Plaintext and the Additional Authenticated Data (AAD).

In case of SEAS, the SM Body SHALL be the Ciphertext as defined by the symmetric encryption algorithm, given by the cipher suite, calculated over the Plaintext. The SM Tag SHALL be the Signature defined by the cipher suite calculated over Ciphertext and AAD.

The Plaintext and the AAD for OSCOAP are defined in Section 5.2.2.

5. CoAP Message Protection

This section presents how OSCOAP protects individual CoAP messages including payload, options and header fields, as well as request-response message exchanges, using the Object-Security option (Section 3) and the Secure Message format (Section 4).

The basic idea is that the significant parts of an unprotected CoAP message - including payload, certain header field and options - are protected using the Secure Message format and sent in a CoAP message with the Object-Security option, in what we then call a "protected" CoAP message. As much a possible of the CoAP message should be protected, but not all CoAP header fields or options can be encrypted and integrity protected, because some are intended to be read or changed by an intermediary node, see Section 6.1 and Section 6.2.

The use of OSCOAP is signaled with the Object-Security option. Endpoints supporting the Object-Security option MUST verify the SM as described in this section before accepting a message as valid. An endpoint receiving a CoAP request with the Object-Security option MUST respond with a CoAP message with the Object-Security option.

The differences between Encryption and Integrity Protection vs Integrity Protection Only is described below. Encryption and Integrity Protection SHALL be used by default.

5.1. Integrity Protection Only

5.1.1. Protected CoAP message formatting

The protected CoAP message is formatted as an ordinary CoAP message, with the following Header, Options and Payload based on the unprotected CoAP message:

- The CoAP header SHALL be the same as the unprotected CoAP message.
- The CoAP options SHALL consist of the same options as the unprotected CoAP message, and the Object-Security option.
- If the unprotected CoAP message has no Payload then the Object-Security option SHALL contain the SM. If the unprotected CoAP message has Payload, then the Object-Security option SHALL be empty and the Payload of the CoAP message SHALL be the SM.

5.1.2. Secure Message formatting

The SM Header, Body and Tag are specified in Section 4.1 and Section 4.2.

The Authenticated Data SHALL consist of the following data, in this order:

- the SM Header;
- the two first bytes of the CoAP header (including Version and Code) with Type and Token Length bits set to 0;
- all CoAP options present which are marked as IP in Figure 2 (Section 6.2), in the order as given by the option number (each Option with Option Header including delta to previous IP-marked Option which is present);
- the CoAP Payload (if any); and
- the Transaction Identifier of the associated CoAP Request, if the message is a CoAP Response (see Section 4.1).

5.1.3. Integrity Protection and Verification

A CoAP endpoint protecting a CoAP message with the Object-Security option using a cipher suite for integrity protection only SHALL generate a protected CoAP message and SM based on the unprotected CoAP message as described in Section 5.1.1 and Section 5.1.2. In addition, the sending endpoint SHALL process the Sequence Number as described in Section 7.

A CoAP endpoint receiving a message containing the Object-Security option SHALL first recreate the Authenticated Data as described in Section 5.1.2, and then verify the SM Tag as defined by the cipher suite associated to the Context Identifier. In addition, the receiving endpoint SHALL process the Sequence Number as described in Section 7.

5.2. Encryption and Integrity Protection

5.2.1. Protected CoAP message formatting

The protected CoAP message is formatted as an ordinary CoAP message, with the following Header, Options and Payload based on the unprotected CoAP message:

- The CoAP header SHALL be the same as the unprotected CoAP message.

- The CoAP options SHALL consist of the unencrypted options of the unprotected CoAP message (those not marked as E in Figure 2 (Section 6.2)), and the Object-Security option. The options shall be formatted as in a CoAP message (each Option with Options Header including delta to previous unencrypted Option).

- If the unprotected CoAP message has no Payload then the Object-Security option SHALL contain the SM. If the unprotected CoAP message has Payload, then the Object-Security option SHALL be empty and the Payload of the CoAP message SHALL be the SM.

5.2.2. Secure Message formatting

The SM Header, Body and Tag are specified in Section 4.1 and Section 4.2.

The Additional Authenticated Data SHALL consist of the following data, in this order:

- the SM Header;

- the two first bytes of the CoAP header (including Version and Code) with Type and Token Length bits set to 0;

- all CoAP options present which are marked as IP but not marked as E in Figure 2 (Section 6.2), in the order as given by the option number (each Option with Option Header including delta to previous IP-marked Option which is present); and

- the Transaction Identifier of the associated CoAP Request, if the message is a CoAP Response (see Section 4.1).

The Plaintext SHALL consist of the following data, formatted as a CoAP message without Header consisting of:

- all CoAP Options present which are marked as E in Figure 2 (see Section 6.2), in the order as given by the Option number (each Option with Option Header including delta to previous E-marked Option); and

- the CoAP Payload, if present, and in that case prefixed by the one-byte Payload Marker (0xFF).

5.2.2.1. Encryption and Decryption

A CoAP endpoint protecting a CoAP message with the Object-Security option using a cipher suite for encryption and integrity protection SHALL generate a protected CoAP message and SM based on the

unprotected CoAP message as described in Section 5.2.1 and Section 5.2.2. In addition, the sending endpoint SHALL process the Sequence Number as described in Section 7.

A CoAP endpoint receiving a message containing the Object-Security option SHALL recreate the Additional Authenticated Data as described in Section 5.1.2 and verify the integrity of, and decrypt the message as defined by the cipher suite associated to the Context Identifier. In addition, the receiving endpoint SHALL process the Sequence Number as described in Section 7.

6. Protected CoAP Message Fields

The CoAP payload SHALL be integrity protected. The CoAP payload SHOULD be encrypted by default.

How CoAP Options and Header Fields shall be protected is described in the remainder of this section.

6.1. Protected CoAP Header Fields

This section describes which CoAP header fields are encrypted or integrity protected end-to-end in OSCOAP.

The CoAP Message Layer parameters, Type and Message ID, as well as Token and Token Length may be changed by a proxy and thus SHALL neither be integrity protected nor encrypted.

The Version and Code fields SHALL be integrity protected, see security considerations.

6.2. Protected CoAP Options

This section describes which CoAP options are encrypted and integrity protected, if present in the unprotected CoAP message.

All CoAP options SHALL be encrypted by default, unless intended to be read by an intermediate node; and SHALL be integrity protected, unless intended to be changed by an intermediate node.

However, some special considerations are necessary because CoAP defines certain legitimate proxy operations, because the security information itself may be transported as an option, and because different processing is performed depending on whether encryption is applied or not.

The details are presented in Section 6.2.1 and Section 6.2.2, and summarized in Figure 2.

No.	C	U	N	R	Name	Format	Length	E	IP
1	x			x	If-Match	opaque	0-8	x	x
3	x	x	-		Uri-Host	string	1-255		a
4				x	ETag	opaque	1-8	x	x
5	x				If-None-Match	empty	0	x	x
6		x	-		Observe	uint	0-3		
7	x	x	-		Uri-Port	uint	0-2		a
8				x	Location-Path	string	0-255	x	x
11	x	x	-	x	Uri-Path	string	0-255	x	b
12					Content-Format	uint	0-2	x	x
14		x	-		Max-Age	uint	0-4		
15	x	x	-	x	Uri-Query	string	0-255	x	b
17	x				Accept	uint	0-2	x	x
20				x	Location-Query	string	0-255	x	x
23	x	x	-		Block2	uint	0-3		
27	x	x	-		Block1	uint	0-3		
28			x		Size2	uint	0-4	x	x
35	x	x	-		Proxy-Uri	string	1-1034		i
39	x	x	-		Proxy-Scheme	string	1-255		i
60			x		Size1	uint	0-4	x	x

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt, IP=Integrity Protect.

Figure 2: Protected CoAP options in OSCOAP

CoAP options marked "i" indicate that they are used as invariants in the authenticated data (AD/AAD) as described in Section 6.2.1.1 and Section 6.2.1.2.

In case of Integrity Protection Only, options marked with "a" and "b" are composed into a URI as described in Section 6.2.1.2 and included as invariant in the Proxy-Uri option in the Authenticated Data.

In case of Encryption and Integrity Protection, options marked "a" are composed into a URI as described in Section 6.2.2 and included as the Proxy-Uri option in the Additional Authenticated Data. (Options marked "b" are included in the Plaintext.)

6.2.1. Integrity Protection

CoAP options which are not intended to be changed by an intermediate node MUST be integrity protected.

- CoAP options of the unprotected message which are Safe-to-Forward SHALL be integrity protected. See Figure 2.

Note: The Object-Security option in itself is Safe-to-Forward but is added to the protected message.

CoAP options which are intended to be modified by a proxy can be divided into two categories, those that are intended to change in a predictable way, and those which are not. The following options are of the latter kind and SHALL NOT be integrity protected:

- Max-Age, Observe, Block1, Block2: These options may be modified by a proxy in a way that is not predictable for client and server.

The remaining options may be modified by a proxy, but when they are, the change is predictable. Therefore it is possible to define "invariants" which can be integrity protected.

6.2.1.1. Proxy-Scheme

A Forward Proxy is intended to replace the URI scheme with the content of the Proxy-Scheme option. The Proxy-Scheme option is defined in this memo to be an invariant with respect to the following processing

- If there is a Proxy-Scheme present in the unprotected message, then the client SHALL integrity protect the Proxy-Scheme option.

- If there is no Proxy-Scheme option present the client SHALL include the Proxy-Scheme option in the authenticated data (AD/AAD) set to the URI scheme. (The sent message does not include the Proxy-Scheme option.)

- The server SHALL insert the Proxy-Scheme option with the name of the URI scheme the message was received in the authenticated data (AD/AAD).

6.2.1.2. Uri-*

For options related to URI of resource (Uri-Host, Uri-Port, Uri-Path, Uri-Query, Proxy-Uri) a Forward Proxy is intended to replace the Uri-* options with the content of the Proxy-Uri option.

The Proxy-Uri option is defined in this memo to be an invariant with respect to the following processing (applied to Integrity Protection only, for Encryption see next section):

- If there is a Proxy-Uri present, then the client MUST integrity protect the Proxy-Uri option and the Uri-* options MUST NOT be integrity protected.

- If there is no Proxy-Uri option present, then the client SHALL compose the full URI from Uri-* options according to the method described in section 6.5 of [RFC7252]. The Authenticated Data contains the following options, modified compared to what is sent:
- All Uri-* options removed
- A Proxy-Uri option with the full URI included
- The server SHALL compose the URI from the Uri-* options according to the method described in section 6.5 of [RFC7252]. The so obtained URI is placed into a Proxy-Uri option, which is included in the Authenticated Data.

6.2.2. Encryption

All CoAP options MUST be encrypted, except the options below which MUST NOT be encrypted:

- Max-Age, Observe, Block1, Block2, Proxy-Uri, Proxy-Scheme: This information is intended to be read by a proxy.
- Uri-Host, Uri-Port: This information can be inferred from destination IP address and port.
- Object-Security: This is the security-providing option.

In the case of encryption, the Proxy-Uri of the Additional Authenticated Data MUST only contain Uri-Host and Uri-Port and MUST NOT contain Uri-Path and Uri-Query because the latter options are not necessarily available to a Forward Proxy.

7. Replay Protection and Freshness

In order to protect from replay of messages and verify freshness of responses, a CoAP endpoint using object security SHALL maintain Sequence Numbers (SEQs) of sent and received Secure Messages (see Section 4.1), associated to the respective security context identified with the Context Identifier (CID).

7.1. Replay Protection

An endpoint SHALL maintain a SEQ for each security context it uses to receive messages, and one SEQ for each security context for protecting sent messages. Depending on use case, an endpoint MAY maintain a sliding receive window for Sequence Numbers in received messages associated to each CID, equivalent to the functionality described in section 4.1.2.6 of [RFC6347].

Before composing a new message a sending endpoint SHALL step the SEQ of the associated CID. However, if the Sequence Number counter wraps, the endpoint must first acquire a new CID and associated security context/key(s). The latter is out of scope of this memo.

A receiving endpoint SHALL verify that the Sequence Number received in the SM Header is greater than the Sequence Number of the associated CID (or within the sliding window and not previously received) and update the SEQ (window) accordingly.

7.2. Freshness

OSCOAP is a challenge-response protocol, where the response is verified to match a prior request by including the unique transaction identifier TID (concatenation of SEQ and CID) of the request in the integrity calculation of the response message.

If a CoAP server receives a request with the Object-Security option, then the authenticated data (AD or AAD) of the response SHALL include the TID of the request as described in Section 5.1.2 and Section 5.2.2.

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 authenticated data (AD or AAD) as described in Section 5.1.2 and Section 5.2.2.

8. Security Considerations

In scenarios with proxies, gateways, or caching, DTLS only protects data hop-by-hop meaning that these intermediary nodes can read and modify information. The trust model where all participating 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 weak.

DTLS protects the entire CoAP message including Header, Options and Payload, whereas OSCOAP protects the payload and message fields described in Section 6.1 and Section 6.2. The cost for DTLS providing this protection is the overhead in e.g. additional messages, processing, memory incurred by the DTLS Handshake protocol, which can be omitted in use cases where key establishment can be provided by other means.

CoAP specifies how messages should be acknowledged on message layer. The CoAP message layer, however, cannot be protected by application layer security end-to-end 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 object security solution, since message layer is excluded from that.

The CoAP Header field Code needs to be integrity protected end-to-end. For example, if a malicous man-in-the-middle would replace the client requested GET with a DELETE, this must be detected by the server. The CoAP Header field Version needs also to be integrity protected to prevent from potential cross-version attacks, such as bidding-down.

Blockwise transfers as defined [I-D.ietf-core-block] cannot be protected with application layer security end-to-end because the Block1/Block2 options may be changed in an unpredictable way by an intermediate node.

However, it is possible to define end-to-end block options analogous to Block1 and Block2 which are safe-to-forward, integrity protected and not supposed to be changed by intermediate devices. With such an option each individual block can be securely verified by the receiver, retransmission securely requested etc. Since the blocks are enumerated sequentially and carry information about last block, when all blocks have been securely received, this proves that the entire message has been securely transferred.

The Observe option cannot be integrity protected since it is allowed to change in an unpredictable way. But since message sequence numbers are integrity protected a client
can verifies that a GET response has not been received before.

The use of sequence numbers for replay protection introduces the problem related to wrapping of the counter. The alternatives also have issues: very constrained devices may not be able to support accurate time or 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.

This specification needs to be complemented with a procedure whereby the client and the server establish the keys used for wrapping and unwrapping the Secure Message. One way to address key establishment is to assume that there is a trusted third party which can support

client and server, such as the Authorization Server in [I-D.ietf-ace-actors]. The Authorization Server may, for example, authenticate the client on behalf of the server, or provide cryptographic keys or credentials to the client and/or server which can be use to derive the keys used in the Secure Message exchange. Similarly, the Authorization Server may, on behalf of the server, notify the client of server supported ciphers, in order to facilitate the usage of OSCOAP in deployments with multiple supported cryptographic algorithms.

The security contexts required are different for different cipher suites. For an AEAD or SEAS it is required to have a unique Initialization Vector for each message, for which the Sequence Number is used. The Initialization Vector SHALL be the concatenation of a Salt (4 bytes unsigned integer) and the Sequence Number. The Salt SHOULD be established between sender and receiver before the message is sent, to avoid the overhead of sending it in each message. For example, the Salt may be established by the same means as keys are established.

9. Privacy Considerations

End-to-end integrity protection provides certain privacy properties, e.g. protection of communication with sensor and actuator from manipulation which may affect the personal sphere. End-to-end encryption of payload and certain CoAP options provides additional protection as to the content and nature of the message exchange.

The headers sent in plaintext allow for example matching of CON and ACK (CoAP Message Identifier), matching of request and response (Token). Plaintext options could also reveal information, e.g. lifetime of measurement (Max-age), or that this message contains one data point in a sequence (Observe).

10. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[this document]" with the RFC number of this specification.

The following entry is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	Object-Security	[[this document]]

This document registers the following value in the CoAP Content Format registry established by [RFC7252].

Media Type: application/oscon

Encoding: -

Id: 70

Reference: [this document]

11. 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 new versions of the draft.

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>.

[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>.

[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>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

12.2. Informative References

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-02 (work in progress), September 2015.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-09 (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-cose-msg]
Schaad, J. and B. Campbell, "CBOR Encoded Message Syntax", draft-ietf-cose-msg-05 (work in progress), September 2015.

[I-D.seitz-ace-core-authz]
Seitz, L., Selander, G., and M. Vucinic, "Authorization for Constrained RESTful Environments", draft-seitz-ace-core-authz-00 (work in progress), June 2015.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[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>.

[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>.

Appendix A. COSE Profile of SM

This section defines a profile of the 05-version of COSE [I-D.ietf-cose-msg] complying with the Secure Message format (see Section 4) and supporting the two scopes of object security OSCOAP and OSCON (Appendix C). In the last subsection we elaborate on possible optimizations.

- The "COSE_MSG" top level object as defined in COSE corresponds to the Secure Message object.
- The "msg_type" parameter corresponds to the Secure Message type, as defined in Section 4.2. Depending on the use case, this field can take the values msg_type_mac, msg_type_signed or msg_type_encryptData.
- The "Header" field of the COSE object corresponds to the Header field of the Secure Message.
  - The "protected" field includes:
    - the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see Section 4.1).
  - The "unprotected" field is empty.

A.1. Integrity Protection Only

When Integrity Protection only needs to be provided, the Secure Message object corresponds to a COSE_MSG with msg_type equal to msg_type_signed (COSE_Sign) or msg_type_mac (COSE_mac).

The Externally Supplied Data ("external_aad" field), as defined in Section 4.1 of [I-D.ietf-cose-msg] include the Authenticated Data as defined in Section 5.1.2 with the exception of SM Header and CoAP Payload.

A.1.1. COSE_Sign

A COSE_MSG of type COSE_Sign is a Secure Message if its fields are defined as follows (see example in Appendix B.2).

The "Headers" field of COSE_Sign as defined in Appendix A.

The "payload" field contains the CoAP Payload (if any).

The "signatures" array contains one "COSE_signature" item. The "Headers" field of the COSE_signature object is defined as follows:

- The "protected" field includes:
  - the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);
- The "unprotected" field is empty.

The "signature" field contains the computed signature value as described in Section 4.2 of [I-D.ietf-cose-msg].

A Secure Message with digital signature and Detached Content corresponds to COSE_sign with "Headers" and "signatures" fields; i.e. no "payload" field.

A.1.2. COSE_mac

A COSE_MSG of type COSE_mac is a Secure Message if its fields are defined as follows (see example in Appendix B.1).

The "Headers" field of COSE_mac as defined in Appendix A.

The "payload" field contains the CoAP Payload (if any).

The "tag" field contains the MAC value, computed as defined in Section 6.1 of [I-D.ietf-cose-msg].

The "recipients" array contains one "COSE_recipient" item (section 5 of [I-D.ietf-cose-msg]). The "COSE_recipient" item contains one "COSE_encrypt_fields" object. The "Headers" field of the COSE_encrypt_fields object is defined as follows:

- The "protected" field includes:
  - the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);
- The "unprotected" field is empty.

A Secure Message with MAC and Detached Content corresponds to a COSE_sign with "Headers", "recipients" and "tag" fields; i.e. no "payload" field.

A.2. Encryption and Integrity Protection: COSE_enveloped

When Encryption and Integrity Protection need to be provided, the Secure Message object corresponds to a COSE_MSG with msg_type equal to msg_type_enveloped (COSE_enveloped).

The Additional Authenticated Data ("Enc_structure") as described is Section 5.3 of [I-D.ietf-cose-msg] is defined in Section 5.2.2: * the "protected" parameters includes the SM Header; * the "external_aad" includes the other fields (CoAP Version, Code, Options to integrity protect and TID).

The plain text, as mentioned in Sections 5.3 and 5.4 of [I-D.ietf-cose-msg] is defined in Section 5.2.2 and contains CoAP Options to encrypt and the CoAP Payload.

A COSE_MSG of type COSE_enveloped [I-D.ietf-cose-msg] is a Secure Message if its fields are defined as follows (see example in Appendix B.3).

The "Headers" field of COSE_encrypt_fields item as defined in Appendix A.

The "ciphertext" field is encoded as a nil type, following the specifications in Section 5.1 of [I-D.ietf-cose-msg].

The "recipients" array contains one "COSE_recipient" item (Section 5.1 of [I-D.ietf-cose-msg]). The "COSE_recipient" item contains one "COSE_encrypt_fields" object. The "Headers" field of the COSE_encrypt_fields object is defined as follows:

- The "protected" field includes:
  - the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);
- The "unprotected" field is empty.

The "ciphertext" field of the COSE_encrypt_fields object contains the encrypted plain text, as defined in section 5 of [I-D.ietf-cose-msg].

A.3. COSE Optimizations

For constrained environments it is important that the message expansion due to security overhead is kept at a minimum.

This section lists potential optimizations of COSE [I-D.ietf-cose-msg] for the purpose of reducing message size and improving performance in constrained node networks. The message sizes resulting from the first four optimizations are presented in Appendix B (as "modified COSE").

1. The first improvement proposed is to flatten the structure of the COSE_msg, following the Encrypted COSE structure defined in

Section 5.2 of [I-D.ietf-cose-msg]. In fact, there is little need to support multiple signatures or recipients in the use cases targeting the most constrained devices. Two different structures inspired by the COSE_encryptData are defined: COSE_ip and COSE_en. COSE_ip is used for the Integrity Protection Only use case (Section 5.1), COSE_en is used for Encryption (Section 5.2).

2. In general, the security context defines uniquely the cipher suite, and hence the "alg" parameter of COSE_msg can be removed.

3. The "unprotected" field is not used since it is assumed that all parameters should be protected when possible. Thus the "Headers" structure can be flattened into a "protectedHeader" field, containing the "cid" parameter and the "seq" parameter.

4. Analogous to other key values, one-byte keys/labels can be assigned to the new parameters defined in this document and cipher suites adapted to constrained device processing. For example: "cid" = 11 and "seq" = 12.

5. Digitally signed messages have the largest absolute overhead due to the size of the signature (see Appendix B.2 and Appendix B.4). Whereas certain MACs can be securely truncated, signatures cannot. Signature schemes with message recovery allow some remedy since they allow part of the message to be recovered from the signature itself and thus need not be sent. The effective size of the signature could in this way be considerably reduced, which would have a large impact on the message size (compare size of signature and total overhead in Figure 5 and Figure 6). A valuable optimization would thus be to support signature schemes with message recovery.

Combining the first 4 points, the resulting structures and their fields are defined as follows: COSE_ip top level object corresponds to the Secure Message object.

- The "msg_type" parameter takes a new value, msg_type_integrityprotection=5.

- The "protectedHeader" field, analogous to the "protected" field of the "Headers", includes:

  * the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);

  * the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see Section 4.1).

- The "payload" field (as described in Appendix A.1.1 and Appendix A.1.2).

- The "tag" field (as described in Appendix A.1.1 and Appendix A.1.2).

COSE_en top level object corresponds to the Secure Message object.

- The "msg_type" parameter takes a new value, msg_type_encryption=6.

- The "protectedHeader" field, analogous to the "protected" field of the "Headers", includes:

  * the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);

  * the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see Section 4.1).

- The "ciphertext" field (as described in Appendix A.2).

- The "tag" field contains the tag value in case Integrity Protection is also provided.

Appendix B. Comparison of message sizes

This section gives some examples of overhead incurred with the current proposal for COSE at the time of writing [I-D.ietf-cose-msg]. Message sizes are also listed for a modified version of COSE implementing some of the optimizations described in Appendix A.3 and for a lower bound CBOR encoding of the Secure Message with structure [seq, cid, body, tag].

Motivated by the use cases, there are four different kinds of protected messages that need to be supported: message authentication code, digital signature, authenticated encryption, and symmetric encryption + digital signature. The latter is relevant e.g. for proxy-caching and publish-subscribe with untrusted intermediary (see Appendix D.2). The sizes estimated for selected algorithms are detailed in the subsections.

The size of the header is shown separately from the size of the MAC/ signature. An 8-byte Context Identifier and a 3-byte Sequence Number are used throughout all examples, with these value:

- cid: 0xa1534e3c5fdc09bd

- seq: 0x112233

For each scheme, we indicate the fixed length of these two parameters ("seq+cid" column) and of the tag ("MAC"/"SIG"/"TAG"). The "Total Size" column shows the total Secure Message size, while the "Overhead" column is calculated from the previous columns following this equation:

Overhead = Total Size - (MAC + seq+cid)

This means that overhead incurring from CBOR encoding is also included in the Overhead count.

To make it easier to read, COSE objects are represented using CBOR's diagnostic notation rather than a binary dump.

B.1. MAC Only

This example is based on HMAC-SHA256, with truncation to 16 bytes.

The object in COSE encoding gives:

```
[
  3,                                          # msg_type
  h'a201046373657143112233',                  # protected:
                                                {1: 4,
                                                "seq": h'112233'}
  {},                                         # unprotected
  h'',                                        # payload
  MAC,                                        # truncated 16-byte MAC
  [                                           # recipients
    [                                         # recipient structure
      h'',                                    # protected
      {1:-6, "cid":h'a1534e3c5fdc09bd'},      # unprotected
      h''                                     # ciphertext
    ]
  ]
]
```

The COSE object encodes to a total size of 53 bytes.

In the modified version of COSE defined in Appendix A.3, the equivalent COSE object would be:

```
[
  5,                                          # msg_type
  h'a20b48a1534e3c5fdc09bd0c43112233',        # protected:
                                                {11:h'a1534e3c5fdc09bd',
                                                 12:h'112233'}
  h'',                                        # payload
  MAC                                         # truncated 16-byte MAC
]
```

This modified COSE object encodes to a total size of 37 bytes.

The low-bound CBOR encoding of this same object is encoded by:

```
[
  h'112233',                                  # seq
  h'a1534e3c5fdc09bd',                        # cid
  h'',                                        # payload
  MAC                                         # truncated 16-byte MAC
]
```

This object encodes to a total size of 32 bytes.

Figure 3 summarizes these results.

Scheme	seq+cid	MAC	Total Size	Overhead
COSE	11 B	16 B	53 bytes	26 bytes
mod-COSE	11 B	16 B	37 bytes	10 bytes
bound	11 B	16 B	32 bytes	5 bytes

Figure 3: Comparison of COSE, modified COSE and CBOR lower bound for HMAC-SHA256.

B.2. Signature Only

This example is based on ECDSA, with a signature of 64 bytes.

The object in COSE encoding gives:

```
[
  1,                                          # msg_type
  h'a16373657143112233',                      # protected:
                                                {"seq": h'112233'}
  {},                                         # unprotected
  h'',                                        # payload
  [                                           # signatures
    [                                         # signature structure
      h'a201266363696448a1534e3c5fdc09bd',    # protected:
                                                {1: -7,
                                              "cid":h'a1534e3c5fdc09bd'}
      {},                                     # unprotected
      SIG                                     # 64-byte signature
    ]
  ]
]
```

The COSE object encodes to a total size of 100 bytes.

In the modified version of COSE defined in Appendix A.3, the equivalent COSE object would be:

```
[
  5,                                          # msg_type
  h'a20b48a1534e3c5fdc09bd0c43112233',        # protected:
                                                {11:h'a1534e3c5fdc09bd',
                                                 12:h'112233'}
  h'',                                        # payload
  SIG                                         # 64-byte signature
]
```

The COSE object encodes to a total size of 86 bytes.

The low-bound CBOR encoding of this same object is encoded by:

```
[
  h'112233',                                  # seq
  h'a1534e3c5fdc09bd',                        # cid
  h'',                                        # payload
  SIG                                         # 64-byte signature
]
```

This object encodes to a total size of 81 bytes.

Figure 4 summarizes these results.

Scheme	seq+cid	SIG	Total Size	Overhead
COSE	11 B	64 B	100 bytes	25 bytes
mod-COSE	11 B	64 B	86 bytes	11 bytes
bound	11 B	64 B	81 bytes	6 bytes

Figure 4: Comparison of COSE, modified COSE and CBOR lower bound for 64 byte ECDSA signature.

B.3. Authenticated Encryption with Additional Data (AEAD)

This example is based on AES-128-CCM-8.

It is assumed that the IV is generated from the Sequence Number and some previously agreed upon Salt. This means it is not required to explicitly send the whole IV in the message.

The object in COSE encoding gives:

```
[
  2,                                          # msg_type
  h'a201046373657143112233',                  # protected:
                                                {1: 4,
                                                "seq": h'112233'}
  {},                                         # unprotected
  TAG,                                        # 8byte authentication tag
  [                                           # recipients
    [                                         # recipient structure
      h'',                                    # protected
      {1:-6, "cid":h'a1534e3c5fdc09bd'},      # unprotected
      h''                                     # ciphertext
    ]
  ]
]
```

The COSE object encodes to a total size of 44 bytes.

In the modified version of COSE defined in Appendix A.3, the equivalent COSE object would be:

```
[
  6,                                          # msg_type
  h'a20b48a1534e3c5fdc09bd0c43112233',        # protected:
                                                {11:h'a1534e3c5fdc09bd',
                                                 12:h'112233'}
  h'',                                        # ciphertext
  TAG                                         # 8byte authentication tag
]
```

The modified COSE object encodes to a total size of 29 bytes.

The low-bound CBOR encoding of this same object is encoded by:

```
[
  h'112233',                                  # seq
  h'a1534e3c5fdc09bd',                        # cid
  h'',                                        # ciphertext
  TAG                                         # 8byte authentication tag
]
```

This object encodes to a total size of 24 bytes.

Figure 5 summarizes these results.

Scheme	seq+cid	TAG	Total Size	Overhead
COSE	11 B	8 B	44 bytes	25 bytes
mod-COSE	11 B	8 B	29 bytes	10 bytes
bound	11 B	8 B	24 bytes	5 bytes

Figure 5: Comparison of COSE, modified COSE and CBOR lower bound for AES-CCM.

B.4. Symmetric Encryption with Asymmetric Signature (SEAS)

This example is based on AES-128-CTR and ECDSA with 64 bytes signature. COSE requires this to be a nested encapsulation of one object into another, here illustrated with a digitally signed AEAD protected object.

The object in COSE encoding gives:

```
[
  1,                                          # msg_type
  h'a16373657143112233',                      # protected:
                                                {"seq": h'112233'}
  {},                                         # unprotected
  h'85024ba2010a6373657143112233a04081834
  0a201256363696448a1534e3c5fdc09bd40',       # payload:
                                          [2,
                                      h'a2010a6373657143112233',
                                      {}, h', [[h'',
                                      {1: -6,
                                           "cid": h'a1534e3c5fdc09bd'
                                           }, h'']]]
  [                                           # signatures
    [                                         # signature structure
      h'a201266363696448a1534e3c5fdc09bd',    # protected:
                                                {1: -7,
                                                "cid":h'a1534e3c5fdc09bd'}
      {},                                     # unprotected
      SIG                                     # 64-byte signature
    ]
  ]
]
```

The COSE object encodes to a total size of 134 bytes.

In the modified version of COSE defined in Appendix A.3, the equivalent COSE object would be:

```
[
  6,                                          # msg_type
  h'a20b48a1534e3c5fdc09bd0c43112233',        # protected:
                                                {11:h'a1534e3c5fdc09bd',
                                                 12:h'112233'}
  h'',                                        # ciphertext
  SIG                                         # 64-byte signature
]
```

This modified COSE object encodes to a total size of 86 bytes.

The low-bound CBOR encoding of this same object is encoded by:

```
[
  h'112233',                                  # seq
  h'a1534e3c5fdc09bd',                        # cid
  h'',                                        # ciphertext
  SIG                                         # 64-byte signature
]
```

This object encodes to a total size of 81 bytes.

Figure 6 summarizes these results.

Scheme	seq+cid	SIG	Total Size	Overhead
COSE	11 B	64 B	134 bytes	59 bytes
mod-COSE	11 B	64 B	86 bytes	11 bytes
bound	11 B	64 B	81 bytes	6 bytes

Figure 6: Comparison of nested AES-CCM within ECDSA (COSE) and combined AES-ECDSA (modified COSE and CBOR lower bound).

Appendix C. Object Security of Content (OSCON)

In this section we define how to only protect the payload/content of individual messages using the Secure Message format (Section 4) to comply with the requirements 1 and 2 in Section 2. This is referred to as Object Security of Content (OSCON).

Note that by only protecting the content of a message it may be verified by multiple recipients. For example, in the case of a proxy that supports caching, a recent response for a certain resource can be cached and used to serve multiple clients. Or, in a publish-subscribe setting, multiple subscribers can be served the same publication. The use of content protection also decouples the binding to the underlying transfer protocol, so the same protected content object can be freely move between CoAP, HTTP, BlueTooth or whatever application layer protocol.

The use of OSCON is signaled with the Content-Format/Media Type set to application/oscon (Section 10). Since the actual format of the content which is protected is lost, that information needs to be added to the message header or known to the recipient.

The sending endpoint SHALL wrap the Payload, and the receiving endpoint unwrap the Payload in the SM format as described in this section. A CoAP client MAY request a response in the OSCON format by setting the option Accept to application/oscon.

In case of cipher suite for integrity protection only, the Authenticated Data SHALL be the concatenation of the SM Header and the CoAP Payload. If case of cipher suite for both encryption and integrity protection, then the AAD SHALL be the SM Header and the

Plaintext SHALL be the CoAP Payload. By default, cipher suites for encryption and integrity protection SHALL be used.

The SM SHALL be protected (encrypted) and verified (decrypted) as described in Section 5.1.3 (Section 5.2.2.1), including replay protection as described in Section 7.1.

Whereas in OSCOAP, the Context Identifier of the SM Header (Section 4.1) typically identifies the sending party, with OSCON (Appendix C) the Context Identifier may well identify the sender and resource.

C.1. Security Considerations of OSCON

OSCON (Appendix C) only protects payload and only gives replay protection (not freshness of response), but allows additional use cases such as point to multi-point interactions including publish-subscribe, reverse proxies and proxy caching of responses. In case of symmetric keys the receiver does not get data origin authentication, which requires a digital signature using a private asymmetric key.

OSCON SHALL NOT be used in cases where CoAP header fields (such as Code or Version) or CoAP options need to be integrity protected. The request for a response in OSCON using the CoAP option Accept set to "application/oscon" is not secured since OSCON does not integrity protect any options. Hence the exchange of OSCON request-response messages is vulnerable to a man-in-the-middle attack where response is exchanged for another response, but since there is replay protection only messages with higher sequence numbers will be accepted.

Blockwise transfers in CoAP as defined in [I-D.ietf-core-block] can be applied with OSCON, i.e. the entire payload is encapsulated in a Secure Message which is partitioned into blocks which are sent with unprotected CoAP. The receiver is able to verify the integrity of the payload but only after the last block containing the signature/MAC is received, and if the verification fails the entire message needs to be resent. However, if the verification succeeds, then the transmission in OSCON has less computational and packet overhead since only one signature/MAC was generated and sent. As CoAP blockwise transfer with OSCON is prone to Denial of Service attacks, it should only be used for exchanges where this threat can be mitigated, for example within a local area network where link-layer security is activated.

Appendix D. Examples

This section gives examples of how to use the Object-Security option and the message formats defined in this memo.

D.1. CoAP Message Protection

This section illustrates Object Security of CoAP (OSCOAP). The message exchange assumes there is a security context established between client and server. One key is used for each direction of the message transfer. The intermediate node detects that the CoAP message contains an OSCOAP object (Object-Security option is set) and thus forwards the message as it cannot serve a cached response.

D.1.1. Integrity Protection of CoAP Message Exchange

Here is an example of a PUT request/response message exchange passing an intermediate node protected with the Object-Security option. The example illustrates a client closing a lock (PUT 1) and getting a confirmation that the lock is closed. Code, Uri-Path and Payload of the request and Code of the response are integrity protected (and other message fields, see Section 6.1 and Section 6.2).

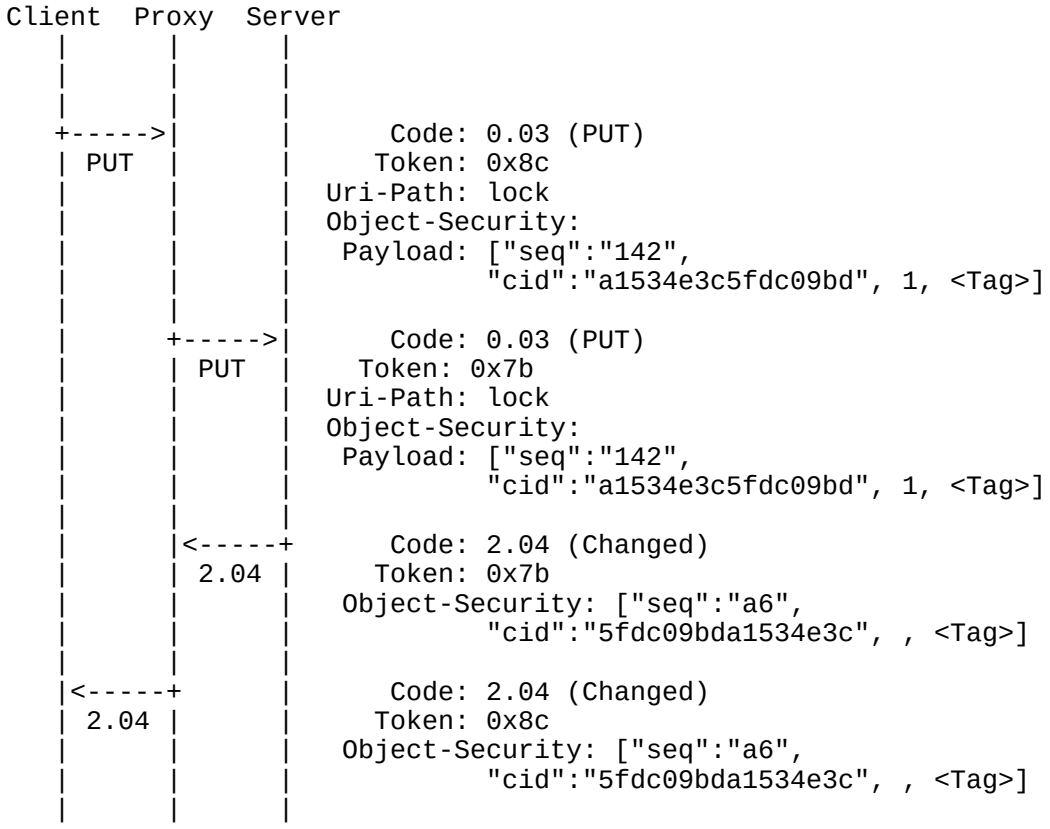


Figure 7: CoAP PUT protected with OSCOAP

Since the request message (PUT) supports payload, the OSCOAP object is carried in the CoAP payload. Since the response message (Changed) does not supports payload the Object-Security option carries the OSCOAP object.

The Header contains Sequence Number ("seq":"a6") and Context Identifier ("cid":"5fdc09bda1534e3c"), the latter is an identifier indicating which security context was used to integrity protect the message, and may be used as an identifier for a secret key or a public key. (It may e.g. be the hash of a public key.)

The server and client can verify that the Sequence Number has not been received and used with this key before. With OSCOAP, the client additionally verifies the freshness of the response, i.e. that the response message is generated as an answer to the received request message (see Section 7).

This example deviates from encryption by default (see Section 8) just to illustrate the case of Integrity Protection only. If there is no compelling reason why the CoAP message should be in plaintext, then it MUST be encrypted.

D.1.2. Additional Encryption of CoAP Message

Here is an example of a GET request/response message exchange passing an intermediate node protected with the Enc option. The example illustrates a client requesting a blood sugar measurement resource (GET /glucose) and receiving the value 220 mg/dl. Uri-Path and Payload are encrypted and integrity protected. Code is integrity protected only (see Section 6.1 and Section 6.2).

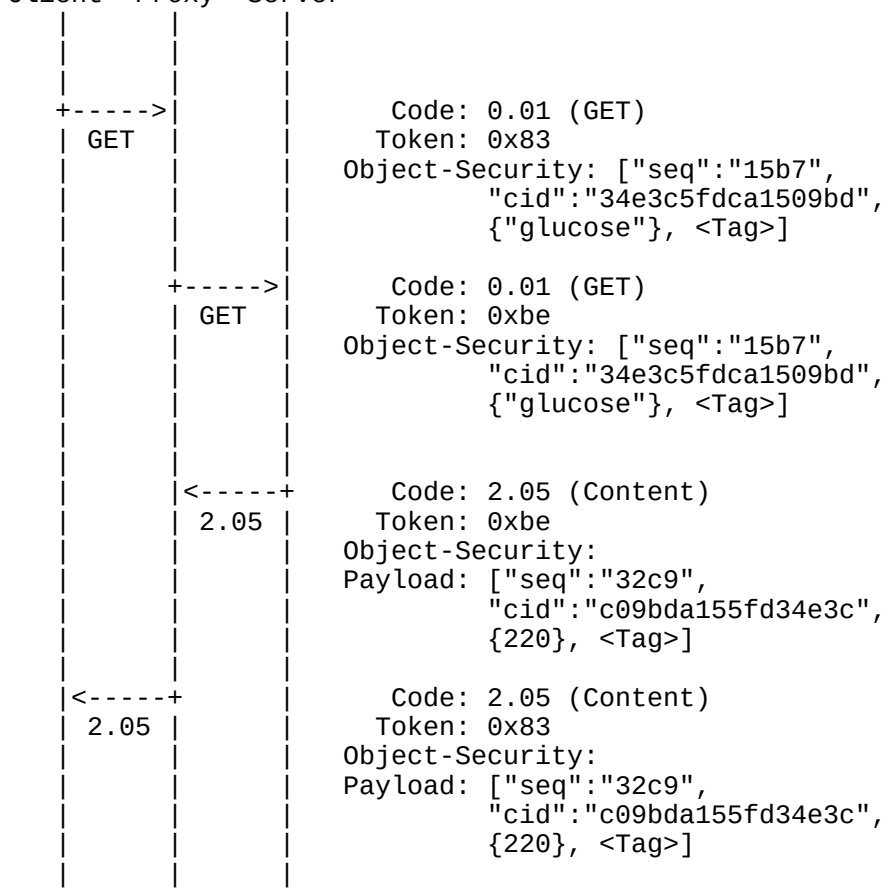


Figure 8: CoAP GET protected with OSCOAP. The bracket { ... } indicates encrypted data.

Since the request message (GET) does not support payload, the OSCOAP object is carried in the Object-Security option. Since the response message (Content) supports payload, the Object-Security option is empty and the OSCOAP object is carried in the payload.

The Context Identifier is a hint to the receiver indicating which security context was used to encrypt and integrity protect the message, and may be used as an identifier for the AEAD secret key. One key is used for each direction of the message transfer.

The server and client can verify that the Sequence Number has not been received and used with this key before, and the client additionally verifies the freshness of the response, i.e. that the response message is generated as an answer to the received request message (see Section 7).

D.2. Payload Protection

This section gives examples that illustrate Object Security of Content (OSCON), see Appendix C). The assumption here is that only the intended receiver(s) has the relevant security context related to the resource. In case of a closed group of recipients of the same object, e.g. in Information-Centric Networking or firmware update distribution, it may be necessary to support symmetric key encryption in combination with digital signature.

D.2.1. Proxy Caching

This example outlines how a proxy forwarding request and response of one client can cache a response whose payload is a OSCON object, and serve this response to another client request, such that both clients can verify integrity and non-replay.

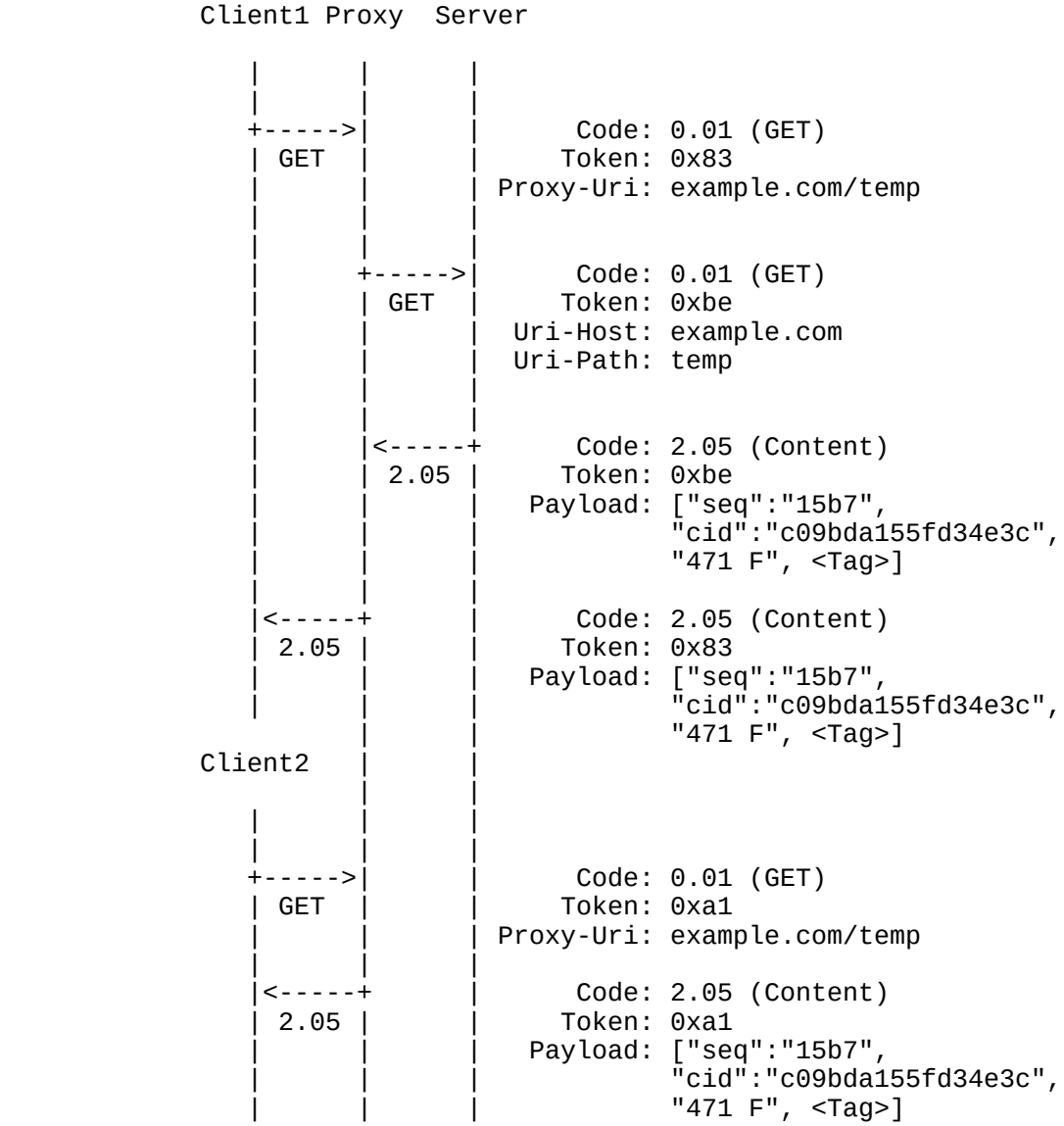


Figure 9: Proxy caching protected with Object Security of Content (OSCON)

D.2.2. Publish-Subscribe

This example outlines a publish-subscribe setting where the payload is encrypted, integrity and replay protected end-to-end between Publisher and Subscriber. The example applies for example to closed

user groups of a single data source and illustrates a subscription registration and a later publication of birch pollen count of 300 per cubic meters. The PubSub Broker can define the Observe count arbitrarily (as could any intermediary node, even in OSCOAP), but cannot manipulate the Sequence Number without being possible to detect.

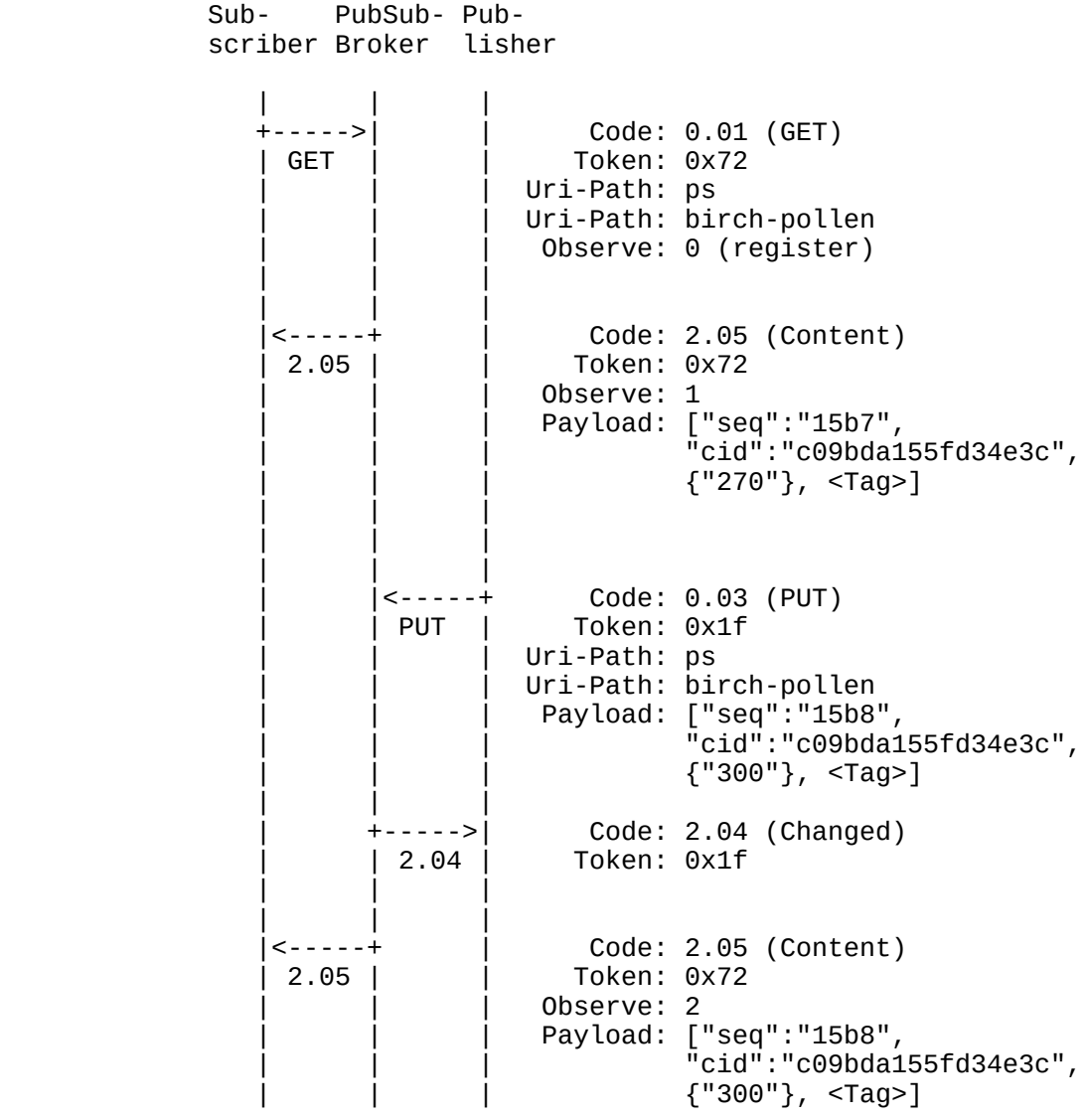


Figure 10: Publish-subscribe protected with OSCON. The bracket { ... } indicates encrypted data.

This example deviates from encryption by default (see Section 8) just to illustrate Integrity Protection only in the case of OSCON. If there is no compelling reason why the payload should be in plaintext, then encryption MUST be used.

D.2.3. Transporting Authorization Information

This example outlines the transportation of authorization information from a node producing (Authorization Server, AS) to a node consuming (Resource Server, RS) such information. Authorization information may for example be an authorization decision with respect to a Client (C) accessing a Resource to be enforced by RS, see e.g. [I-D.ietf-ace-actors] or [I-D.seitz-ace-core-authz]. Here, C is clearly not trusted with modifying the information, but may need to be involved in mediating the authorization information to the RS, for example, because AS and RS does not have direct connectivity. So end-to-end security is required and object security ("access tokens") is the natural candidate.

This example considers the authorization information to be encapsulated in a OSCON object, generated by AS. How C accesses the OSCON object is out of scope for this example, it may e.g. be using CoAP. C then requests RS to configure the authorization information in the OSCON object by doing POST to /authz-info. This particular resource has a default access policy that only new messages signed by AS are authorized. RS thus verifies the integrity and sequence number by using the existing security context for the AS, and responds accordingly, a) or b), see Figure 11.

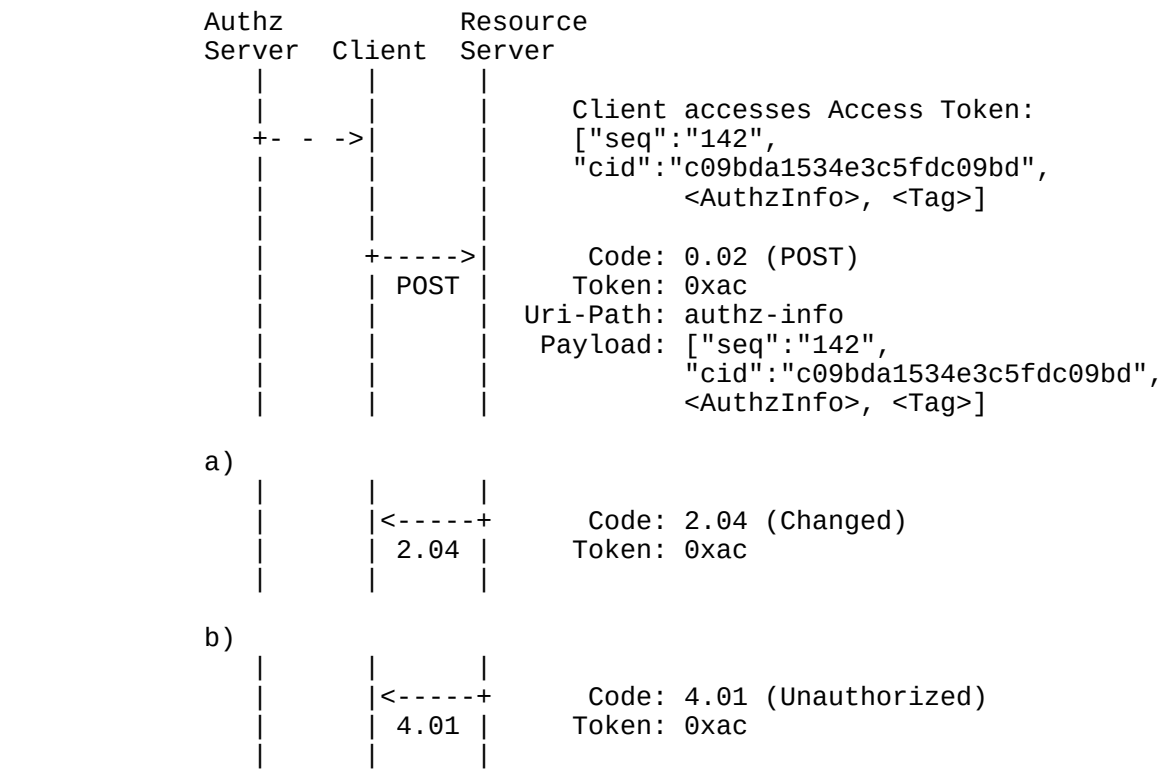


Figure 11: Protected Transfer of Access Token using OSCON

Authors' Addresses

Goeran Selander
Ericsson
Farogatan 6
Kista 16480
Sweden

Email: goran.selander@ericsson.com

John Mattsson
Ericsson
Farogatan 6
Kista 16480
Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson
Farogatan 6
Kista 16480
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: December 22, 2015

B. Silverajan
Tampere University of Technology
T. Savolainen
Nokia
June 20, 2015

CoAP Communication with Alternative Transports
draft-silverajan-core-coap-alternative-transports-08

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.

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 . . . 14
A.1. Transport information as part of the URI authority . . . 14
A.1.1. Usage of DNS records 15
A.2. Making CoAP Resources Available over Multiple Transports 15
A.3. Transport as part of a 'service:' URL scheme 18
Authors' Addresses 18

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 definied 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 end point 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] specifices 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.tschofenig-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 [I-D.ietf-core-observe].
[I-D.tschofenig-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 nodespossess 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:

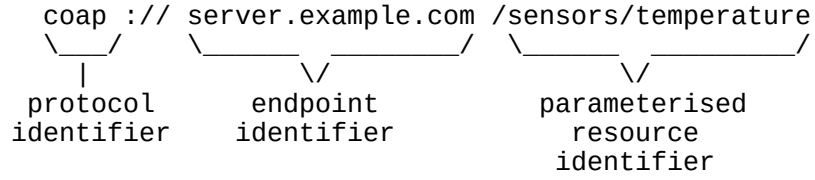


Figure 1: The CoAP URI format

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:

- coap+tcp://[2001:db8::1]:5683/sensors/temperature for using CoAP over TCP
- coap+tls://[2001:db8::1]:5683/sensors/temperature for using CoAP over TLS
- coaps+sctp://[2001:db8::1]:5683/sensors/temperature for using CoAP over DTLS over SCTP
- coap+sms://0015105550101/sensors/temperature for using CoAP over SMS with the endpoint identifier being a telephone subscriber number
- coaps+sms://0015105550101/sensors/temperature for using CoAP over DTLS over SMS with the endpoint identifier being a telephone subscriber number
- coap+ws://www.example.com/sensors/temperature for using CoAP over WebSockets

- 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 [I-D.ietf-6lo-btle]. 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, January 2005.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

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-6lo-btle]
Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", draft-ietf-6lo-btle-13 (work in progress), May 2015.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-16 (work in progress), December 2014.

[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-09 (work in progress), June 2015.

[I-D.savolainen-core-coap-websockets]
Savolainen, T., Hartke, K., and B. Silverajan, "CoAP over WebSockets", draft-savolainen-core-coap-websockets-04 (work in progress), March 2015.

[I-D.tschofenig-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-tschofenig-core-coap-tcp-tls-04 (work in progress), June 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, March 1997.

[RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates and Service: Schemes", RFC 2609, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

[RFC5092] Melnikov, A. and C. Newman, "IMAP URL Scheme", RFC 5092, November 2007.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

[RFC6568] Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, April 2012.

[RFC6733] Fajardo, V., Arkko, J., Loughney, J., and G. Zorn, "Diameter Base Protocol", RFC 6733, October 2012.

[RFC7390] Rahman, A. and E. Dijk, "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, October 2014.

[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:

- coap-at://tcp-server.example.com/sensors/temperature

- 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-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.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:

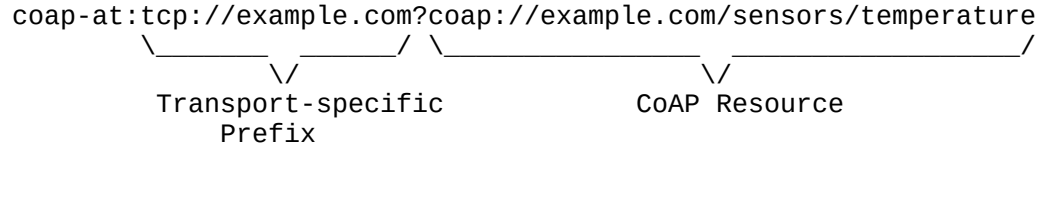


Figure 2: Prefixing a CoAP URI with TCP transport

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:

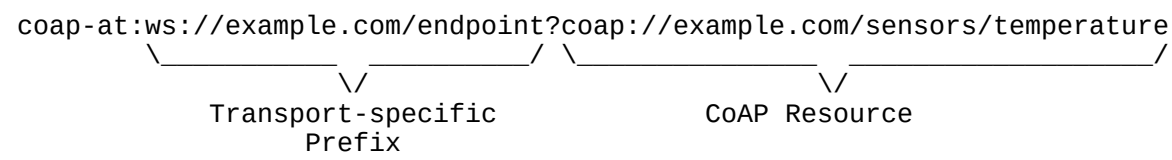


Figure 3: Prefixing a CoAP URI with WebSocket transport

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

Teemu Savolainen
Nokia
Hermiankatu 12 D
FI-33720 Tampere
Finland

Email: teemu.savolainen@nokia.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: March 11, 2016

B. Silverajan
Tampere University of Technology

September 8, 2015

CoAP Protocol Negotiation
draft-silverajan-core-coap-protocol-negotiation-01

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 March 11, 2016.

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 -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 configuratons 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.tschofenig-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 specifices 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 negotations 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.savolainen-core-coap-websockets]
Savolainen, T., Hartke, K., and B. Silverajan, "CoAP over WebSockets", draft-savolainen-core-coap-websockets-04 (work in progress), March 2015.

[I-D.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-08 (work in progress), June 2015.

[I-D.tschofenig-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-tschofenig-core-coap-tcp-tls-04 (work in progress), June 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 -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
Internet Draft
Intended status: Standards track
Expires: April 2016

A. Bhattacharyya
S. Bandyopadhyay
A. Pal
T. Bose
Tata Consultancy Services Ltd.

October 15, 2015

CoAP option for no server-response
draft-tcs-coap-no-response-option-12

Abstract

There can be M2M scenarios where responses from server against requests from client might be considered 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 simultaneously updating a bulk of resources or updating a resource with a very high frequency. CoAP already provides a non-confirmable (NON) mode of message exchange where the server end-point does not respond with ACK. However, obeying the request/response semantics, the server end-point responds back with a status code indicating "the result of the attempt to understand and satisfy the request".

This draft introduces a header option for CoAP called ’No-Response’. Using this option the client explicitly tells the server to suppress responses against the particular request. This option also provides granular control to enable suppression of a particular class or a combination of response-classes. This option may be effective for both unicast and multicast requests. Present draft also discusses 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/1id-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 April 15, 2016.

Copyright Notice

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
3. Exemplary application scenarios.....7
   3.1. Frequent update of geo-location from vehicles to backend..7
   3.2. Multicasting actuation command from a handheld device to a group of appliances.....8
      3.2.1. Using granular response suppression.....9
4. Miscellaneous aspects.....9
   4.1. Re-using Tokens.....9
   4.2. Taking care of congestion.....10
   4.3. Handling No-Response option for a reverse proxy.....11
5. Example.....11
   5.1. Using No-Response with PUT.....11
   5.2. Using No-Response with POST.....12
      5.2.1. POST updating a fixed target resource.....12

5.2.2. POST updating through query-string.....13
6. IANA Considerations.....14
7. Security Considerations.....15
8. Acknowledgments.....15
9. References.....15
9.1. Normative References.....15
9.2. Informative References.....15

1. Introduction

This draft proposes a new header option for Constrained Application Protocol (CoAP) [RFC7252] called 'No-Response'. This option enables the client end-point to explicitly express its disinterest in receiving responses back from the server end-point. This option allows all classes of response by default. Fine grain control to suppress responses of a particular class or a combination of response classes is also possible.

Along with the technical details this draft presents some practical application scenarios which should bring out the usefulness of this option.

Wherever, in this draft, it is mentioned that a request from client is with No-Response the intended meaning is that the client expressed 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 systems may benefit in the following aspects:

* Reduction in network clogging due to effective reduction of the overall traffic.

* Reduction in server-side loading by relieving the server from responding to each request when not necessary.

* Reduction in battery consumption at the constrained end-point.

* Reduction in overall communication cost.

1.2. Terminology

The terms used in this draft 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 this option are given in Table 1.

Number	C	U	N	R	Name	Format	Length	Default
284			X		No-Response	uint	0-1	0

Table 1: Option Properties

This option is a request option. It is Elective and Non-Repeatable.

Note: Since CoAP maintains a clear separation between the request/response and the messaging layer, this option does not have any dependency on the type of message (confirmable/ non-confirmable). However, 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 have any significance if piggybacked responses are triggered. But, in case the server responds with a separate response (which, may be, the client does not care about) then this option can be useful. Suppressing the separate response reduces one additional traffic 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. The following table provides a ’ready-reference’ on possible applicability of this option for all the four REST methods. This table is prepared in view of the type of possible interactions foreseen so far.

Method Name	Remarks on applicability
GET	This SHOULD NOT be used with GET under usual circumstances when the client requests the contents of a resource. However, this option may be useful for special GET requests. At present only one such application is identified which is the 'cancellation' procedure for 'Observe'. Observe-cancellation requires a client to issue a GET request with Observe option set to 'deregister' (1). Since, in this case, the server response does not contain any payload the client MAY express its disinterest in server responses.
PUT	Suitable for frequent updates (particularly in NON mode) on existing resources. Might not be useful when PUT creates a new 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 5.2.2 for an example).
DELETE	Deletion is usually a permanent action and the client MAY want to ensure that the deletion actually happened. MAY NOT be applicable.

Table 2: Suggested applicability of No-Response for different REST methods

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.

Note: Section 3.7 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. Client does not have any knowledge about that. 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 3.2.

This option is defined as a bit-map (Table 3) to achieve granular suppression.

Value	Binary Representation	Description
0	<empty>	Allow all responses.
2	00000010	Suppress 2.xx responses.
8	00001000	Suppress 4.xx responses.
16	00010000	Suppress 5.xx responses.
127	01111111	Suppress all responses.

Table 3: 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,

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. At present the "CoAP Response Codes" registry (Ref. Section 12.1.2 of [RFC7252]) defines only 2.xx, 4.xx and 5.xx responses.

So, an option value of 22 (binary: 00010110) will effectively suppress all currently defined response codes.

3. To suppress all possible responses: The maximum reserved response code for CoAP is 7.31 (Ref. Section 12.1 of [RFC7252]). So, setting bit positions 0-6 will suppress all responses according to the combination operation defined in point 2 above. Hence, the value to block all present and possible future responses is: 2**7 - 1 = 127.

Implementation Note: When No-Response is used with empty or 0 value in a request the client end-point SHOULD cease listening to response against the particular request. On the other hand, opening up 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 update will be a success or a failure. Thus, for example, if the client decides to open up the response for errors (4.xx & 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 successful. The application designer needs to tackle such situation. For example, while performing frequent updates, the client may strategically interweave requests without No-Response 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.

3. Exemplary application scenarios

This section describes some exemplary user stories which may potentially get benefitted through the use of No-Response option.

3.1. Frequent update of geo-location from vehicles to backend

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 end-point. 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. The update rate is adaptive to the motional-state of the vehicle. If the vehicle moves fast the update rate is high as the position of the vehicle changes rapidly. If the vehicle is static or moves slowly then the update rate is low. This ensures that bandwidth and energy is not consumed unnecessarily. The motional-state of the vehicle is inferred by a

local analytics running on the sensor-gateway which uses the accelerometer data and the rate of change in GPS co-ordinates. The back-end server hosts applications which use the updates for each vehicle and produce necessary information for remote users.

Retransmitting a location co-ordinate which the vehicle has already crossed is not efficient as it adds redundant traffic to the network. So, the updates are done in NON mode. However, given the huge number of vehicles updating frequently, the NON exchange will also trigger huge number of status responses from the backend. Thus the cumulative load on the network will be quite significant.

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 significant load will be reduced. The assumption is that, since the update rate is high, stray losses in geo-locations 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 ([I-D.ietf-core-observe]) with NON notifications. But, in practice, implementing with "Observe" would require lot of book-keeping exercise 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 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. Putting it another way: 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 it is the interest of the data-producer. The 'No-Response' option enables to make classic updates further less resource consuming.

3.2. 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 the smart phone can be programmed to send a multicast request to switch on/ off all the lights of a building. In this case the IP switch application can use No-Response

option along with NON request to reduce the traffic generated due to simultaneous status responses from hundreds of lights.

Thus No-Response helps in reducing overall communication cost and the probability of network clogging in this case.

3.2.1. 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.

4. Miscellaneous aspects

This section further describes few important implementation aspects worth considering while using No-Response. The following discussion does not mandate anything, rather suggests some guidelines for the application developer.

4.1. 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 gap between using the same tokens is not 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. This draft suggests a reuse time for tokens with similar expression as in Section 2.5 of [RFC7390]:

TOKEN_REUSE_TIME = NON_LIFETIME + MAX_SERVER_RESPONSE_DELAY + 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. But for 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 it is the expected delay "over all servers that the client can send a multicast request to". This delay includes the maximum Leisure time period as defined in Section 8.2 of [RFC7252]. [RFC7252] defines a rough lower bound of leisure as:

$$lb\_Leisure = S * G / R$$

(S = estimated response size; R = data transfer rate; G = group size estimate)

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 request with No-Response to the same server endpoint.

4.2. Taking care of congestion

A detail technical discussion on congestion control is out-of-scope of this draft. However, this section of the draft mention certain aspects on congestion control which may help a detail work on congestion control for CoAP as a whole.

If this option is used with NON messages then the interaction becomes completely open-loop. Absence of any feed-back from the server end affects congestion-control mechanism. In this case the communication pattern belongs to the class of low-data volume applications as described in Section 3.1.2 of [RFC5405]. Precisely, it maps to the scenario where the application cannot maintain an RTT estimate. Hence, following [RFC5405], a 3s interval is suggested as the minimum interval between successive updates. However, in case of frequent updates, an application developer MUST interweave

occasional closed-loop exchanges (e.g. NON messages without No-Response or simply CON messages) to get an RTT estimate between the end-points.

4.3. Handling No-Response option for a reverse proxy

A reverse proxy (HTTP to CoAP) MAY translate an incoming HTTP request to a corresponding CoAP request indicating that no response is required based on some application specific requirement. In this case, it is recommended that the HC Proxy SHOULD send an HTTP response with status code 204 (No Content).

5. Example

This section illustrates few examples of exchanges based on the scenario narrated in Section 3.1. Examples for other scenarios can be easily conceived based on these illustrations.

5.1. Using No-Response with PUT

Figure 1 shows a typical request with this option. The depicted scenario occurs when the vehicle#n moves very fast and update rate is high. The 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.

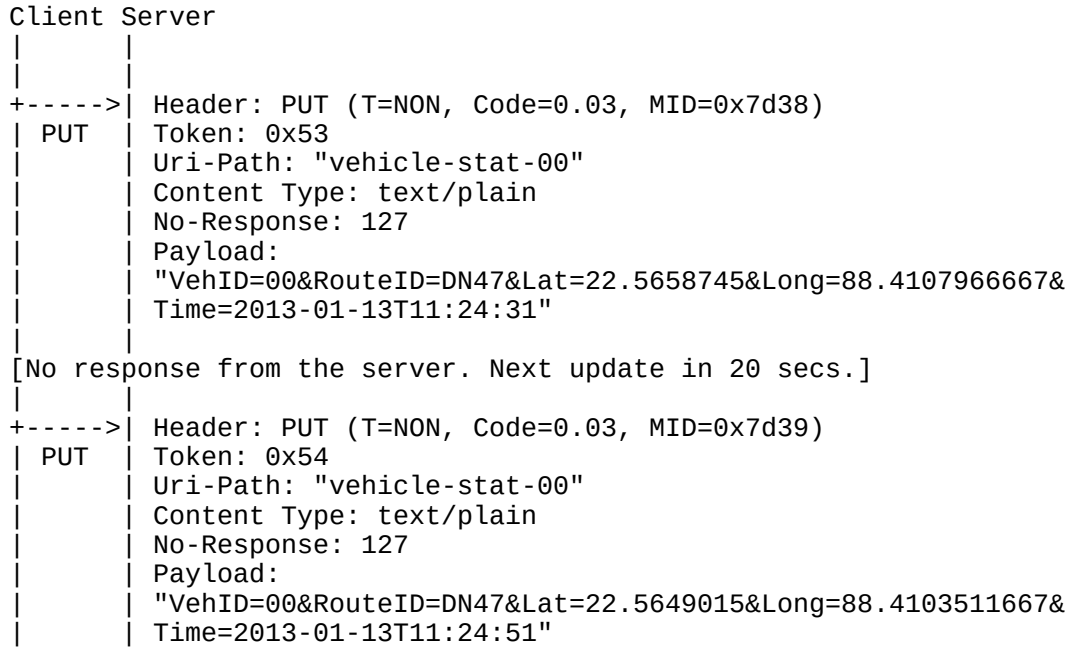


Figure 1: Exemplary unreliable update with No-Response option using PUT.

5.2. Using No-Response with POST

5.2.1. 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.

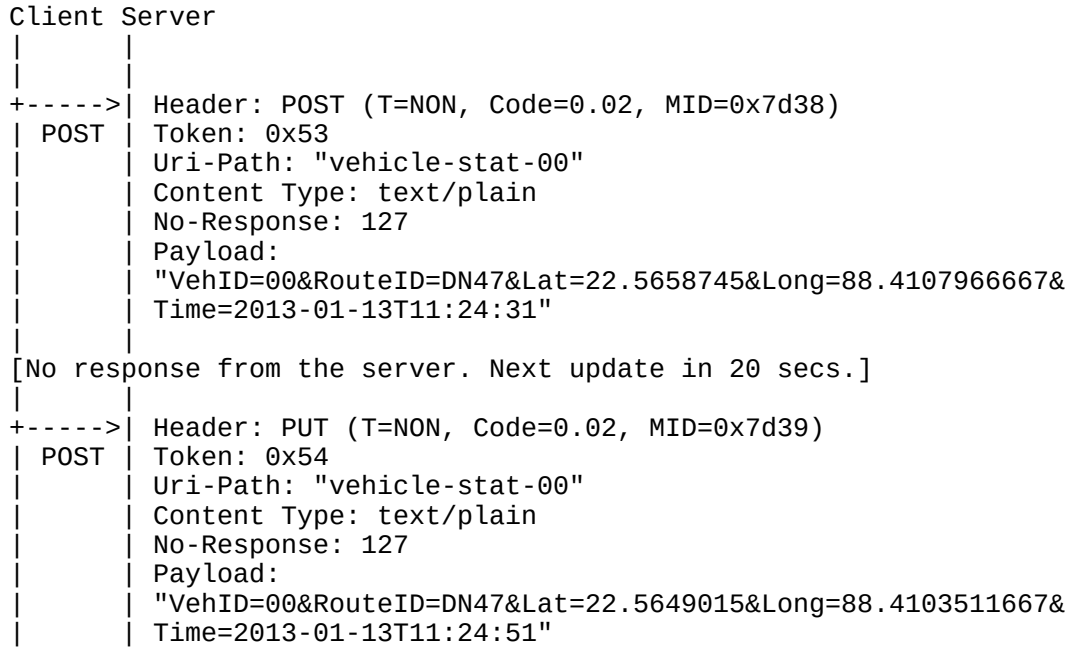


Figure 2: Exemplary unreliable update with No-Response option using POST as the update-method.

5.2.2. POST updating through query-string

It may be possible that the backend infrastructure (as described in Section 3.1) 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.

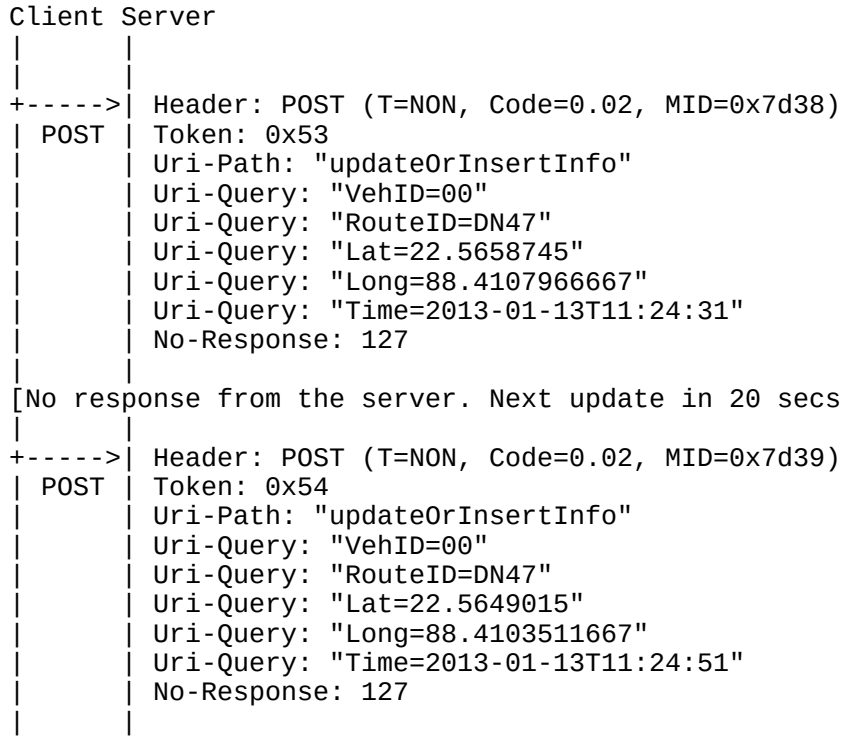


Figure 3: Exemplary unreliable update with No-Response option using POST with a query-string to insert update information to backend database.

6. IANA Considerations

The IANA has assigned number 284 to this option in the CoAP Option Numbers registry:

Number	Name	Reference
284	No-Response	Section 2 of this document

7. 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].

8. Acknowledgments

Thanks to Carsten Bormann, Matthias Kovatsch, Esko Dijk, Bert Greevenbosch, Akbar Rahman and Klaus Hartke for their valuable inputs.

9. References

9.1. Normative References

[RFC7252]

Shelby, Z., Hartke, K. and Bormann, C.,"Constrained Application Protocol (CoAP)", RFC 7252, June, 2014

[I-D.ietf-core-observe]

Hartke, K.,"Observing Resources in CoAP", draft-ietf-core-observe-16, December 30, 2014

[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

9.2. Informative References

[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: Standards Track
Expires: December 12, 2015

C. Bormann, Ed.
Universitaet Bremen TZI
S. Lemay
V. Solorzano Barboza
Zebra Technologies
H. Tschofenig
ARM Ltd.

June 10, 2015

A TCP and TLS Transport for the Constrained Application Protocol (CoAP)
draft-tschofenig-core-coap-tcp-tls-04.txt

Abstract

The Hypertext Transfer Protocol (HTTP) has been designed with TCP as an underlying transport protocol. The Constrained Application Protocol (CoAP), which has been inspired by HTTP, has on the other hand been defined to make use of UDP. 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 that 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 December 12, 2015.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Constrained Application Protocol 3
4. Message Format 5
4.1. Discussion 8
5. Message Transmission 8
6. CoAP URI 9
6.1. coap+tcp URI scheme 9
6.2. coaps+tcp URI scheme 9
7. Security Considerations 10
8. IANA Considerations 10
8.1. Service Name and Port Number Registration 10
8.2. URI Schemes 11
8.3. ALPN Protocol ID 12
9. Acknowledgements 12
10. References 12
10.1. Normative References 12
10.2. Informative References 13
Authors' Addresses 13

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP can be used freely - UDP [RFC0768], or DTLS [RFC6347] over UDP, is a good choice for transferring small amounts of data in 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.

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 both IoT devices work smoothly in these demanding environments, CoAP needs to make use of a different transport protocol, namely TCP [RFC0793] and in some situations even TLS [RFC5246].

The present document document describes a shim header that conveys length information about each CoAP message included. 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 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).

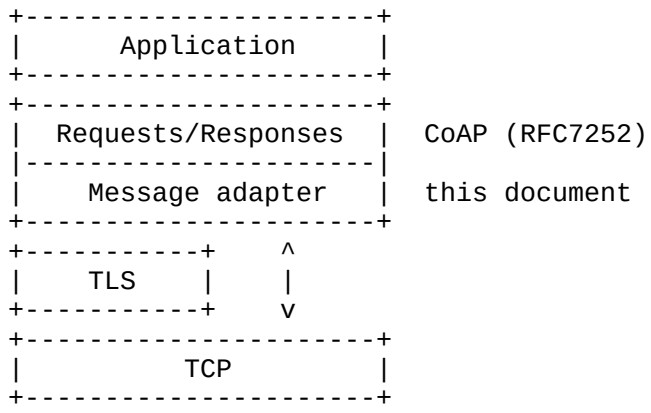


Figure 1: The CoAP over TLS/TCP Protocol Stack

TCP offers features that are not available in UDP and consequently have been provided in the message layer of CoAP. Since TCP offers reliable delivery, there is no need to offer a redundant acknowledgement at the CoAP messaging layer.

Hence, the only message type transported when using CoAP over TCP is the Non-Confirmable message (NON). By nature of TCP, a NON over TCP is still transmitted reliably. 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 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).

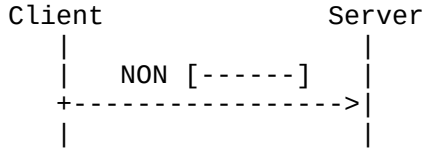


Figure 2: NON Message Transmission over TCP.

As a result of removing the message layer in CoAP over TCP, there is no longer a need to distinguish message types. Since the two-bit field for the message needs to be filled with something, all messages

are marked with the bit combination indicating the NON type (no message layer acknowledgement is expected or even possible). A response is sent back as defined in [RFC7252], as illustrated in Figure 3 (derived from [RFC7252], Figure 6).

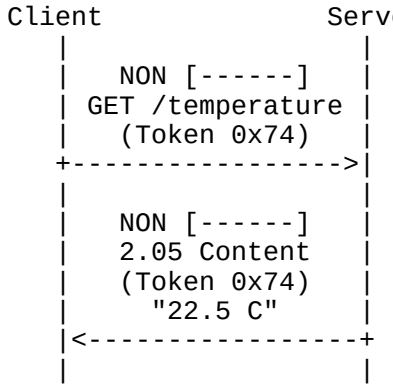


Figure 3: NON Request/Response.

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.

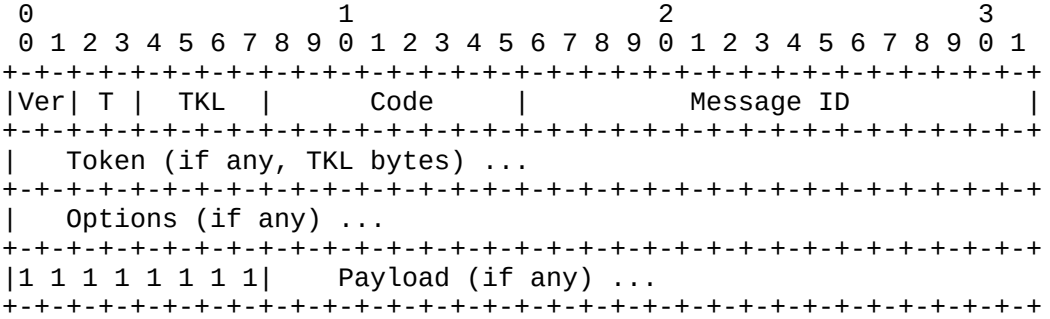


Figure 4: RFC 7252 defined CoAP Message Format.

In a stream oriented transport protocol such as TCP, some other form of delimiting messages is needed. For this purpose, CoAP over TCP introduces a length field. Figure 5 shows a 1-byte shim header carrying length information prepending the CoAP message header.

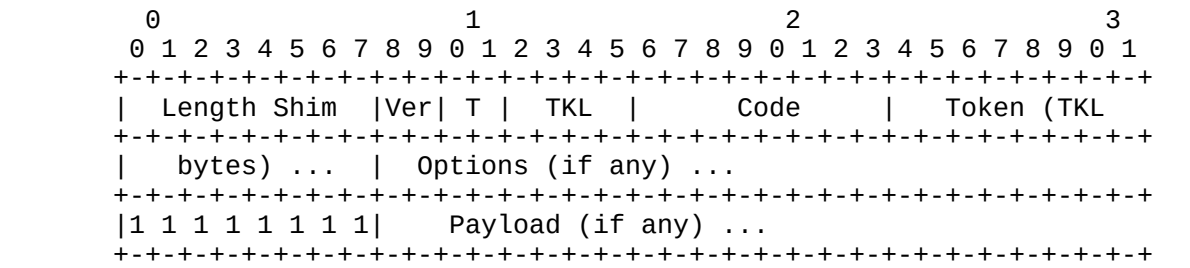


Figure 5: CoAP Header with prepended Shim Header.

-- Alternative L1 --

The 'Message Length' field is a 16-bit unsigned integer in network byte order.

-- Alternative L2 --

The 'Message Length' field starts with an 8-bit unsigned integer. Length encoding follows the same mechanism as "Major type 0" from the CBOR specification [RFC7049]. The length field is indicated by the 5 least significant bits of the byte. Values are used as such:

- between 0b000_00001 and 0b000_10111 (1 to 23) indicates the actual length of the following message
- 0b000_11000 (24) means an additional 8-bit unsigned Integer is appended to the initial length field indicating the total length
- 0b000_11001 (25) means an additional 16-bit unsigned Integer (in network byte order) is appended to the initial length field indicating the total length
- 0b000_11010 (26) means an additional 32-bit unsigned Integer (in network byte order) is appended to the initial length field indicating the total length

The 3 most significant bits in the initial length field are reserved for future use. If a recipient gets a message larger than it can handle, it SHOULD if possible send back a 4.13 in accordance with [RFC7252] section on error code.

-- Common for L1 and L2 Alternatives --

The "length" field provides the length of the subsequent CoAP message (including the CoAP header but excluding this message length field) in bytes. T is always the code for NON (1).

-- Alternative L3 --

The initial byte of the frame contains two nibbles, in a similar way to the CoAP option encoding (Section 3.1 of [RFC7252]). The first nibble is used to indicate the length of the options (including any option delimiter), and the payload (if any); it does not include the Code byte or the Token bytes. The first nibble is interpreted as a 4-bit unsigned integer. A value between 0 and 12 directly indicates the length of the options/payload, in bytes. The other three values have a special meaning:

13: An 8-bit unsigned integer follows the initial byte and indicates the length of options/payload minus 13.

14: A 16-bit unsigned integer in network byte order follows the initial byte and indicates the length of options/payload minus 269.

15: A 32-bit unsigned integer in network byte order follows the initial byte and indicates the length of options/payload minus 65805.

The second nibble of the initial byte indicates the token length.

Example: 01 43 7f is a frame just containing a 2.03 code with the token 7f.

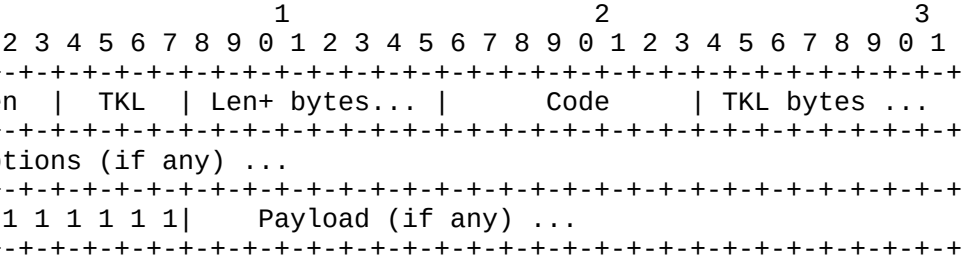


Figure 6: CoAP Header with prepended Shim Header (L3).

-- End L Alternatives

The Message ID is meaningless and thus elided. The semantics of the other CoAP header fields is left unchanged.

4.1. Discussion

One might wish that, when CoAP is used over TLS, then the TLS record layer length field could be used in place of the shim header length. Each CoAP message would be transported in a separate TLS record layer message, making the shim header that includes the length information redundant.

However, RFC 5246 says that "Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType MAY be coalesced into a single TLSPlaintext record, or a single message MAY be fragmented across several records)." While the Record Layer provides length information about the encapsulated application data and handshaking payloads, TLS implementations typically do not support an API interface that would provide access to the record layer delimiting information. An additional problem with this approach is that this approach would remove the potential optimization of packing several CoAP messages into one record layer message, which is normally a way to amortize the record layer and MAC overhead over all these messages.

In summary, we are not pursuing this idea for an optimization.

One other observation is that the message size limitations defined in Section 4.6 of [RFC7252] are no longer strictly necessary. Consenting [how?] implementations may want to interchange messages with payload sizes 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:

- 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.

- large messages might also cause undesired head-of-line blocking.

The general assumption is therefore that the block protocol will continue to be used over TCP, even if 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 elicited a

response yet are canceled; clients are free to transmit the request again once a connection is reestablished.

Furthermore, since TCP is bidirectional, requests can be sent from both the connecting host or the endpoint that accepted the connection. In other words, who initiated the TCP connection has no bearing on the meaning of the CoAP terms client and server, which are relating only to an individual request and response pair.

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, applies to this URI scheme, with the following changes:

- 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, applies to this URI scheme, with the following changes:

- 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).

- 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 Internet of Things deployments where clients are pre-configured to only ever talk with specific servers. [[_1: 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 [RFC4395].

8.3. ALPN Protocol ID

This document requests a value from the "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, and Zach Shelby for their feedback.

10. References

10.1. Normative References

[I-D.ietf-dice-profile] Tschofenig, H. and T. Fossati, "A TLS/DTLS Profile for the Internet of Things", draft-ietf-dice-profile-12 (work in progress), May 2015.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, July 2014.

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-02 (work in progress), July 2014.

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-17 (work in progress), March 2015.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[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, August 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

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
Intended status: Standards Track
Expires: April 18, 2016

P. van der Stok
consultant
A. Bierman
YumaWorks
J. Schoenwaelder
Jacobs University
A. Sehgal
consultant
October 16, 2015

CoAP Management Interface
draft-vanderstok-core-comi-08

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. It is designed to reduce the message sizes, server code size, and application development complexity. The Constrained Application Protocol (CoAP) is used to access management data resources specified in YANG, or SMIv2 converted to YANG. The payload of the CoMI message is encoded in Concise Binary Object Representation (CBOR).

Note

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

Status of This Memo

Table of Contents

1. Introduction 3
1.1. Design considerations 5
1.2. Terminology 5
1.2.1. Tree Diagrams 6
2. CoMI Architecture 6
2.1. RESTCONF/YANG Architecture 10
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 14
4.1.2. Mapping of the 'select' Parameter 14
4.1.3. Retrieval Examples 15
4.2. Data Editing 27
4.2.1. Data Ordering 27
4.2.2. POST 27
4.2.3. PUT 27
4.2.4. PATCH 28
4.2.5. DELETE 32
4.2.6. Editing Multiple Resources 32
4.3. Notify functions 34
4.4. Use of Block 36
4.5. Resource Discovery 37
4.6. Error Return Codes 38
5. Mapping YANG to CoMI payload 39
5.1. YANG Hash Generation 40
5.2. Re-Hash Error Procedure 41
5.3. Reverse Re-Hash Error Procedure 42
5.4. ietf-yang-hash YANG Module 42
5.5. YANG Re-Hash Examples 45
5.5.1. Multiple Modules 46

5.5.2. Same Module 47
5.6. Retrieval of Rehashed Data 48
5.7. YANG Hash in URL 49
6. Mapping YANG to CBOR 50
6.1. High level encoding 50
6.2. Conversion from YANG datatypes to CBOR datatypes 50
7. Error Handling 51
8. Security Considerations 53
9. IANA Considerations 53
10. Acknowledgements 53
11. Changelog 54
12. References 57
12.1. Normative References 57
12.2. Informative References 58
Appendix A. Payload and Server sizes 62
Appendix B. Notational Convention for CBOR data 63
Appendix C. CBOR examples 64
Appendix D. Comparison with LWM2M 67
Appendix E. Hash clash probability 68
Appendix F. Hash clash storage overhead 71
F.1. Server tables 71
F.2. Client tables 71
F.3. Table summary 72
Authors' Addresses 73

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 draft [I-D.ietf-netconf-restconf] describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG [RFC6020]. 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.

The CoRE Management Interface (CoMI) is intended to work in a stateless client-server fashion. CoMI (and also RESTCONF) uses a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips.

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 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. To promote small packets, CoMI uses an additional "data-identifier string-to-number conversion" to minime CBOR payloads and URI length. It is assumed that the managed device is the most constrained entity. The client might be more capable, however this is not necessarily the case.

Currently, small managed devices need to support at least two protocols: CoAP and SNMP [RFC3411]. When the MIB can be accessed with the CoAP 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.

The objective of CoMI is to provide a Function Set that reads and sets values of managed objects in devices to (1) initialize parameter values at start-up, (2) acquire statistics during operation, and (3) maintain nodes by adjusting parameter values during operation.

The goal of CoMI is to provide information exchange in a uniform manner as a first step to the full management functionality as specified in [I-D.ersue-constrained-mgmt].

1.1. Design considerations

CoMI supports discovery of resources, accompanied by reading, writing and notification of resource values. As such it is close to the device management of the Open Mobile Alliance described in [OMA]. A comparison between CoMI and LWM2M management can be found in Appendix D. CoMI supports MIB modules which have been translated once from SMIv2 to YANG, using [RFC6643]. This mapping is read-only so writable SMIv2 objects need to be converted to YANG using an implementation-specific mapping.

The YANG data model contains a lot of information that can be exploited by automation tools and need not be transported in the request messages, ultimately leading to reduced message sizes.

1.2. 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].

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 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. 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.

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.

The following list contains the abbreviations used in this document.

XXXX: TODO, and others to follow.

1.2.1. 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.

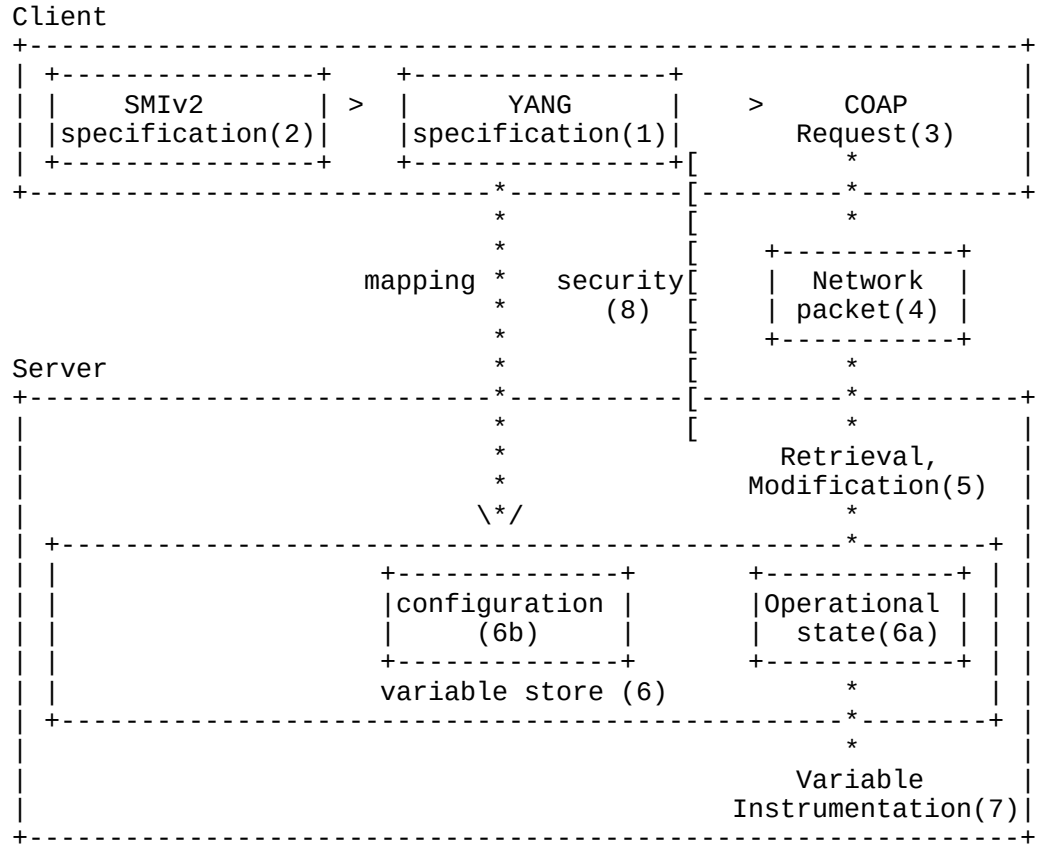


Figure 1: Abstract CoMI architecture

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:

- Equip a constrained node with a management server that provides information about the operational characteristics of the code running in the constrained node.
- The server provides this information in a variable store that contains values describing the performance characteristics and the code parameter values.

- The client receives the performance characteristics on a regular basis or on request.
- The client sets the parameter values in the server at bootstrap and intermittently when operational conditions change.
- 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:

- The names, type, and semantics of the instrumentation variables are standardized.
- The instrumentation variables are described in a standard language.
- The URI of the CoAP request is standardized.
- The format of the packet payload is standardized.
- 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 analying 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/<hash-value>".

(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. CoMI allows access to the same data resources as a RESTCONF server, except that messages are optimized to reduce identifier and payload size.

RESTCONF uses 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). CoMI uses a compression algorithm to reduce the size of the data-node instance identifier (see Section 2.2.

2.2. Compression of data-node instance identifier

The RESTCONF protocol uses the full path of the desired data resource in the target resource URI. The JSON 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 in RESTCONF messages, 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 greatly 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. This encoding in 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 4.5).

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 as YANG hash values (see Section 5.7).

/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 hash values. An EntityTag 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. The only value defined in this document is

'yanghash' to indicate that the YANG Hash numbering scheme defined in this document 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 of Section 5.4 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. If a client requests a node that has been re-hashed, then a rehash error is returned, according to the procedure in Section 5.2.

/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.

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 YANG hash values, as described in Section 5.1. When multiple instances of a list node exist, the instance selection is described in Section 4.1.3.4

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

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 4.4.

There are two query parameters for the GET method. A CoMI server MUST implement the keys 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 selected sub-trees from the target resource

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. Mapping of the ’select’ Parameter

RESTCONF uses the ’select’ parameter to specify an expression which can represent a subset of all data nodes within the target resource [I-D.ietf-netconf-restconf]. This parameter is useful for filtering sub-trees and retrieving only a subset that a managing application is interested in.

However, 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, only a limited subset of the 'select' parameter is used here.

Unlike the RESTCONF 'select' parameter, CoMI does not use object names in "XPath" or "path-expr" format to identify the subset that needs to be filtered. Parsing XML is resource intensive for constrained devices [management] and using object names can lead to large message sizes. Instead, CoMI utilizes the YANG hashes described in Section 5 to identify the sub-trees that should be filtered from a target resource. Using these hashes ensures that a constrained node can identify the target sub-tree without expending many resources and that the messages generated are also efficiently encoded.

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. Retrieval Examples

In all examples the path is expressed in readable names and as a hash value of the name (where the hash value in the payload is expressed as a hexadecimal number, and the hash value in the URL as a base64 number). CoMI payloads use the CBOR format. The CBOR syntax of the YANG payloads is specified in Section 5. The examples in this section use a JSON payload with extensions to approach the permissible CBOR payload. Appendix C shows the CBOR format of some of the examples.

4.1.3.1. 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 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 5.1) 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 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 (YANG hash, 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"
  }
}
```

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 request using the 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"
   }
}
```

The corresponding CBOR code can be found in Appendix C.

4.1.3.2. 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"}
   }
}
```

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 request using the hash values is shown below:

```
REQ: GET example.com/mg/kReR4
   (Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
   0x2445e478 : {
     {0x2283ed40 : "fe80::200:f8ff:fe21:67cf"}:
        {0x3d6915c7 : "00:00::10:01:23:45"}
     ,
     {0x2283ed40 : "fe80::200:f8ff:fe21:6708"}:
        {0x3d6915c7 : "00:00::10:54:32:10"}
     ,
     {0x2283ed40 : "fe80::200:f8ff:fe21:88ee"}:
        {0x3d6915c7 : "00:00::10:98:76:54"}
   }
}
```

4.1.3.3. 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"
        }
      ]
    }
  }
}
```

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/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 following example shows a request for the entire ipNetToPhysicalTable. The payload is a CBOR map composed of two (key , object} pairs, where key and object are CBOR maps, composed of 3 and 5 (identifier, value) pairs respectively.

```
REQ: GET example.com/mg/Gqt28

RES: 2.05 Content (Content-Format: application/cbor)
{
    0x06aaddbc: {
        {
          0x346b3071 : 1,
          0x3650bb64 : "ipv4",
          0x06fd4d91 : "10.0.0.51"}:
        {
          0x26180bcb : "00:00:10:01:23:45",
          0x3d6bbe90 : "2333943",
          0x35ecbb3d : "static",
          0x13038bb5 : "reachable",
          0x09e1fa37 : "active"
        },
        {
          0x346b3071 : 1,
          0x3650bb64 : "ipv4",
          0x06fd4d91 : "9.2.3.4"}:
        {
          0x26180bcb : "00:00:10:54:32:10",
          0x3d6bbe90 : "2329836",
          0x35ecbb3d : "dynamic",
          0x13038bb5 : "unknown",
          0x09e1fa37 : "active"
        }
     }
}
```

The corresponding CBOR code can be found in Appendix C.

4.1.3.4. 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 a YANG hash value. 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 1 instance is requested from the ipNetToPhysicalEntry (from a previous example). The CBOR payload is constructed as before.

```
REQ: GET example.com/mg/Gqt28?keys=1,ipv4,10.0.0.51

RES: 2.05 Content (Content-Format: application/cbor)
{
   0x06aaddbc: {
      {
          0x346b3071 : 1,
          0x3650bb64 : "ipv4",
          0x06fd4d91 : "9.2.3.4"}:
        {
          0x26180bcb : "00:00:10:54:32:10",
          0x3d6bbe90 : "2329836",
          0x35ecbb3d : "dynamic",
          0x13038bb5 : "unknown",
          0x09e1fa37 : "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 col1 { type uint32; }
      }
    }
  }
```

The path identifier for the leaf "col1" is the following string:

```
   /foo-mod:A/B/col1
```

The YANG hash for this identifier string has values: 0x189295aa and YkpWq).

The following string represents the CoMI target resource identifier for the instance of the "col1" leaf with key values "top", 17, "group":

```
/mg/YkpWq?keys="top",17,"group1"
```

4.1.3.5. The 'select' Query Parameter

The select parameter is used along with the GET method to provide a sub-tree filter mechanism. A list of YANG hashes 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.

The following example shows an "ipNetToPhysicalTable" (from a previous example) with 4 instances, using JSON encoding following [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" : 3,
          "ipNetToPhysicalNetAddressType" : "ipv4",
          "ipNetToPhysicalNetAddress" : "9.2.3.4",
          "ipNetToPhysicalPhysAddress" : "00:00:10:54:32:10",
          "ipNetToPhysicalLastUpdated" : "2329836",
          "ipNetToPhysicalType" : "dynamic",
          "ipNetToPhysicalState" : "unknown",
          "ipNetToPhysicalRowStatus" : "active"
        },
        {
          "ipNetToPhysicalIfIndex" : 2,
          "ipNetToPhysicalNetAddressType" : "ipv4",
          "ipNetToPhysicalNetAddress" : "10.24.2.53",
          "ipNetToPhysicalPhysAddress" : "00:00:10:28:19:CA",
          "ipNetToPhysicalLastUpdated" : "2124368",
          "ipNetToPhysicalType" : "static",
          "ipNetToPhysicalState" : "unknown",
          "ipNetToPhysicalRowStatus" : "active"
        },
        {
          "ipNetToPhysicalIfIndex" : 1,
          "ipNetToPhysicalNetAddressType" : "ipv4",
          "ipNetToPhysicalNetAddress" : "192.168.2.12",
          "ipNetToPhysicalPhysAddress" : "00:00:10:29:11:32",
          "ipNetToPhysicalLastUpdated" : "1925384",
          "ipNetToPhysicalType" : "dynamic",
          "ipNetToPhysicalState" : "reachable",
          "ipNetToPhysicalRowStatus" : "active"
        }
      ]
    }
  }
}
```

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/?select=Gqt28(1,ipv4)

RES: 2.05 Content (Content-Format: application/cbor)
{
    0x06aaddbc: {
      {
          0x346b3071 : 1,
          0x3650bb64 : "ipv4",
          0x06fd4d91 : "10.0.0.51"}:
        {
          0x26180bcb : "00:00:10:01:23:45",
          0x3d6bbe90 : "2333943",
          0x35ecbb3d : "static",
          0x13038bb5 : "reachable",
          0x09e1fa37 : "active"
        },
        {
          0x346b3071 : 1,
          0x3650bb64 : "ipv4",
          0x06fd4d91 : "192.168.2.12"}:
        {
          0x26180bcb : "00:00:10:29:11:32",
          0x3d6bbe90 : "1925384",
          0x35ecbb3d : "dynamic",
          0x13038bb5 : "reachable",
          0x09e1fa37 : "active"
        }
   }
}
```

In this example exactly 2 instances are returned as response from the ipNetToPhysicalTable because both those instances match the provided keys.

Supposing there were multiple YANG hashes 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. The following string represents the CoMI target resource identifier when multiple YANG hashes, with their own sets of keys are queried:

```
    /mg/?select=hash1(hash1-key1,hash1-key2,...),hash2(hash2-key1)...
```

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, however it is only allowed for creation of data resources. The same constraints apply as defined in section 3.4.1 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.

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 col1 { 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: (for the CBOR contents see Appendix C.

```
 {
    "B":  {
     { "key1" : "author1",
      "key2" : "book2"}:
      { null : null},
     { "key1" : "author5"} :
       {"counter1" : 4444},
     { "key1" : "newauthor",
      "key2" : "newbook"}:
     { "col1" : 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
```

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
```

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";
    }
  }
}

}
```

The YANG Hash values for this notification are assigned as follows:

```
0x3fe84d89: /example-port:example-port-fault
0x2921ba9e: /example-port:example-port-fault/port-name
0x2d452885: /example-port:example-port-fault/port-fault
0x11287619 (RKHUZ) : /stream
```

}

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"
      }
}
```

Replacing the names by the hash values leads to:

```
REQ: GET example.com/mg/RKHUZ
    (observe option register)

RES: 2.05 Content (Content-Format: application/cbor)
{
    0x3fe84d89 : {
      0x2921ba9e : "0/4/21",
      0x2d452885 : "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. 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 are foreseen for data streaming purposes.

4.5. 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 hash values of the object identifiers to /.well-known/core with rt="core.mg.data". The available objects identified by the hash 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 hash 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 hash 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 Section 5.1. When a collision occurred in the name space of the target server, a rehash is executed as explained in Section 5.2.

4.6. 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.

TODO: assign an error code for a rehash-error.

RESTCONF Status Line	CoAP Status Code
100 Continue	none?
200 OK	2.05
201 Created	2.01
202 Accepted	none?
204 No Content	?
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. Mapping YANG to CoMI payload

A mapping for the encoding of YANG data in CBOR is necessary for the efficient transport of management data in the CoAP payload. Since object names may be rather long and may occur repeatedly, CoMI allows for association of a given object path identifier string value with an integer, called a "YANG hash".

5.1. 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 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.

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 the following section is executed.

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

/ietf-interfaces:interfaces/interface/mtu

Example: the following identifier is 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

5.2. 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 device. 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. 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 are only an issue when they occur at the same server. The client needs to discover any re-hash mappings on a per server basis.

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

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 needs to be retrieved. A client does not need to retrieve the rehash map before retrieving or altering 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) has 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).

5.3. Reverse Re-Hash Error Procedure

A hash collision occurs if two different path identifier strings have the same hash value. If the server has over 30,000 node names in its YANG modules, then the probability of a collision is 10% or higher, see Appendix E. 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.

5.4. 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@2015-06-06.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>

        Editor:   Juergen Schoenwaelder
                  <mailto:j.schoenwaelder@jacobs-university.de>

        Editor:   Anuj Sehgal
                  <mailto:s.anuj@jacobs-university.de>";

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

       Copyright (c) 2015 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-vanderstok-core-comi-08.txt

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.
    revision 2015-09-24 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: CoMI Protocol.";
    }

    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>
```

5.5. 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
```

5.5.1. 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
             }
           ]
        }
     ]
  }
}
```

5.5.2. 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"
             }
           ]
        }
     ]
  }
}
```

5.6. 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.

5.7. 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 Bx is translated into a character Cx 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".

6. Mapping YANG to CBOR

6.1. High level encoding

When encoding YANG variables in CBOR, the CBOR encodings entry is a map, composed of (key, value) pairs. The key is the YANG hash of entry variable, whereas the value contains its value.

For encoding of the variable values, a CBOR datatype is used. Section 6.2 provides the mapping between YANG datatypes and CBOR datatypes.

6.2. Conversion from YANG datatypes to CBOR datatypes

Table 1 defines the mapping between YANG datatypes and CBOR datatypes.

Elements of types not in this table, and of which the type cannot be inferred from a type in this table, are ignored in the CBOR encoding by default. Examples include the "description" and "key" elements. However, conversion rules for some elements to CBOR MAY be defined elsewhere.

YANG type	CBOR type	Specification
int8, int16, int32, int64, uint16, uint32, uint64, decimal64	unsigned int (major type 0) or negative int (mayor type 1)	The CBOR integer type depends on the sign of the actual value.
boolean	either "true" (major type 7, simple value 21) or "false" (major type 7, simple value 20)	
string	text string (major type 3)	
enumeration	unsigned int (major type 0)	
bits	array of text	Each text string contains the

	strings	name of a bit value that is set.
binary	byte string (major type 2)	
empty	null (major type 7, simple value 22)	TBD: This MAY not be applicable to true MIBs, as SNMP may not support empty variables...
union		Similar to the JSON transcription from [I-D.ietf-netmod-yang-json], the elements in a union MUST be determined using the procedure specified in section 9.12 of [RFC6020].
leaf-list	array (major type 4)	The array is encapsulated in the map associated with the YANG variable.
list	map (major type 5) of maps (major type 5)	Map of array element pairs (index, value). Each array element contains a map of associated YANG hash - value pairs.
container	map (major type 5)	The map contains YANG hash - value pairs corresponding to the elements in the container.
smiv2:oid	array of integers	Each integer contains an element of the OID, the first integer in the array corresponds to the most left element in the OID.

Table 1: Conversion of YANG datatypes to CBOR

7. 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].

8. Security Considerations

For secure network management, it is important to restrict access to MIB 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.

9. IANA Considerations

'rt="core.mg.data"' needs registration with IANA.

'rt="core.mg.moduri"' needs registration with IANA.

'rt="core.mg.modset"' needs registration with IANA.

'rt="core.mg.yang-hash"' needs registration with IANA.

'rt="core.mg.yang-stream"' needs registration with IANA.

Content types to be registered:

- application/comi+cbor

10. 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, Zach Shelby, Hannes Tschofenig, Michel Veillette, Michael Verschoor, and Thomas Watteyne. The CBOR encoding borrows extensively from Ladislav Lhotka's description on conversion from YANG to JSON.

This material is based upon work supported by Philips Research, Huawei, and The Space & Terrestrial Communications Directorate (S&TCD); the latter 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 Philips Research, Huawei, or The Space & Terrestrial Communications Directorate (S&TCD).

Juergen Schoenwaelder and Anuj Sehgal were partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

11. Changelog

Changes from version 00 to version 01

- Focus on MIB only
- Introduced CBOR, JSON, removed BER
- defined mappings from SMI to xx
- Introduced the concept of addressable table rows

Changes from version 01 to version 02

- Focus on CBOR, used JSON for examples, removed XML and EXI
- added uri-query attributes mod and con to specify modules and contexts
- Definition of CBOR string conversion tables for data reduction
- use of Block for multiple fragments
- Error returns generalized
- SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- Added security considerations

Changes from version 03 to version 04

- Added design considerations section
- Extended comparison of management protocols in introduction
- Added automatic generation of CBOR tables
- Moved lowpan table to Appendix

Changes from version 04 to version 05

- Merged SNMP access with RESTCONF access to management objects in small devices
- Added CoMI architecture section
- Added RESTCONf NETMOD description
- Rewrote section 5 with YANG examples
- Added server and payload size appendix
- Removed Appendix C for now. It will be replaced with a YANG example.

Changes from version 04 to version 05

- Extended examples with hash representation
- Added keys query parameter text
- Added select query parameter text
- Better separation between specification and instance
- Section on discovery updated
- Text on rehashing introduced
- Elaborated SMI MIB example
- Yang library use described
- use of BigEndian/LittleEndian in Hash generation specified

Changes from version 05 to version 06

- Hash values in payload as hexadecimal and in URL in base64 numbers
- Streamlined CoMI architecture text
- Added select query parameter text
- Data editing optional
- Text on Notify added
- Text on rehashing improved with example

Changes from version 06 to version 07

- reduced payload size by removing JSON hierarchy
- changed rehash handling to support small clients
- added LWM2M comparison
- Notification handling as specified in YANG
- Added Patch function
- Rehashing completely reviewed
- Discover type of YANG name encoding
- Added new resource types
- Read-only servers introduced
- Multiple updates explained

Changes from version 07 to version 08

- Changed YANG Hash algorithm to use module name instead of prefix
- Added rehash bit to allow return values to identify rehashed nodes in the response
- Removed /mg/mod.set resource since this is not needed
- Clarified that YANG Hash is done even for unimplemented objects
- YANG lists transported as CBOR maps of maps
- Adapted examples with more CBOR explanation

- Added CBOR code examples in new appendix
- Possibility to use other than default stream
- Added text and examples for Patch payload
- Repaired some examples
- Added appendices on hash clash probability and hash clash storage overhead

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>.

[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>.

[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-06 (work in progress), October 2015.

[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-patch] Stok, P. and A. Sehgal, "Patch Method for Constrained Application Protocol (CoAP)", draft-vanderstok-core-patch-02 (work in progress), October 2015.

[murmur3] "murmurhash family", Web http://en.wikipedia.org/wiki/MurmurHash.

[murmur-imp] "murmurhash implementation", Web https://code.google.com/p/smhasher/.

12.2. 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>.

[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>.

[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>.

[RFC6650] Falk, J. and M. Kucherawy, Ed., "Creation and Use of Email Feedback Reports: An Applicability Statement for the Abuse Reporting Format (ARF)", RFC 6650, DOI 10.17487/RFC6650, June 2012, <http://www.rfc-editor.org/info/rfc6650>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[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>.

[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>.

[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.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-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/.

[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/.

[coll-prob] Preshing, j., "Hash collision probablilities", Web http://preshing.com/20110504/hash-collision-probabilities, May 2011.

[birthday] Wikipedia, , "Birthday problem", Web https://en.wikipedia.org/wiki/Birthday_problem.

[Erbium] Kovatsch, M., "Erbium Memory footprint for coap-18", Private Communication .

[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.

[I-D.ietf-netconf-yang-patch] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-04 (work in progress), June 2015.

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. Notational Convention for CBOR data

To express CBOR structures [RFC7049], this document uses the following conventions:

A declaration of a CBOR variable has the form:

```
name : datatype;
```

where "name" is the name of the variable, and "datatype" its CBOR datatype.

The name of the variable has no encoding in the CBOR data.

"datatype" can be a CBOR primitive such as:

tstr: A text string (major type 3)

uint: An unsigned integer (major type 0)

map(x,y): A map (major type 5), where each first element of a pair is of datatype x, and each second element of datatype y. A '.' character for either x or y means that all datatypes for that element are valid.

A datatype can also be a CBOR structure, in which case the variable's "datatype" field contains the name of the CBOR structure. Such CBOR structure is defined by a character sequence consisting of first its name, then a '{' character, then its subfields and finally a '}' character.

A CBOR structure can be encapsulated in an array, in which case its name in its definition is preceded by a '*' character. Otherwise the structure is just a grouping of fields, but without actual encoding of such grouping.

The name of an optional field is preceded by a '?' character. This means, that the field may be omitted if not required.

Appendix C. CBOR examples

TODO: inconsistent with examples in text

The two examples in this appendix show the use of the CBOR map. In each example, the JSON code of the text is shown followed by the corresponding CBOR code that MUST be transported.

The first example show the transport of two leaves of a simple container

```
--------------------------JSON example ----------------------
{
  0x021ca491 : {
      0x047c468b : "2014-10-26T12:16:51Z",
      0x1fb5f4f8 : "2014-10-21T03:00:00Z"
   }
}
-----------------------------CBOR code ----------------------
a1                                      # map(1)
   1a 021ca491                          # unsigned(35431569)
   a2                                   # map(2)
      1a 047c468b                       # unsigned(75253387)
      74                                # text(20)
         323031342d31302d32365431323a31363a35315a # "2014-10-26T12:16:51Z"
      1a 1fb5f4f8                       # unsigned(532018424)
      74                                # text(20)
         323031342d31302d32315430333a30303a30305a # "2014-10-21T03:00:00Z"
```

The following figure shows the transport of two YANG list items. A map of length 2 (the 2 list items) is composed of two maps: the key of length 3 and the value of length 5.

```
   --------------------------JSON example ----------------------
   {
       0x06aaddbc: {
           {
             0x346b3071 : 1,
             0x3650bb64 : "ipv4",
             0x06fd4d91 : "10.0.0.51"}:
           {
             0x26180bcb : "00:00:10:01:23:45",
             0x3d6bbe90 : "2333943",
             0x35ecbb3d : "static",
             0x13038bb5 : "reachable",
```

```
            0x09e1fa37 : "active"
          },
          {
            0x346b3071 : 1,
            0x3650bb64 : "ipv4",
            0x06fd4d91 : "9.2.3.4"}:
          {
            0x26180bcb : "00:00:10:54:32:10",
            0x3d6bbe90 : "2329836",
            0x35ecbb3d : "dynamic",
            0x13038bb5 : "unknown",
            0x09e1fa37 : "active"
          }
       }
   }
   ------------------------------CBOR code -----------------------
   a1                                      # map(1)
      1a 06aaddbc                          # unsigned(111861180)
      a2                                   # map(2)
         a3                                # map(3)
            1a 346b3071                    # unsigned(879439985)
            01                             # unsigned(1)
            1a 3650bb64                    # unsigned(911260516)
            64                             # text(4)
               69707634                    # "ipv4"
            1a 06fd4d91                    # unsigned(117263761)
            69                             # text(9)
               31302e302e302e3531          # "10.0.0.51"
         a5                                # map(5)
            1a 26180bcb                    # unsigned(639110091)
            71                             # text(17)
               30303a30303a31303a30313a32333a3435 # "00:00:10:01:23:45"
            1a 3d6bbe90                    # unsigned(1030471312)
            67                             # text(7)
               32333333393433              # "2333943"
            1a 35ecbb3d                    # unsigned(904706877)
            66                             # text(6)
               737461746963                # "static"
            1a 13038bb5                    # unsigned(318999477)
            69                             # text(9)
               726561636861626c65          # "reachable"
            1a 09e1fa37                    # unsigned(165804599)
            66                             # text(6)
               616374697665                # "active"
         a3                                # map(3)
            1a 346b3071                    # unsigned(879439985)
            01                             # unsigned(1)
            1a 3650bb64                    # unsigned(911260516)
```

```
            64                                # text(4)
               69707634                       # "ipv4"
            1a 06fd4d91                       # unsigned(117263761)
            67                                # text(7)
               392e322e332e34                 # "9.2.3.4"
         a5                                   # map(5)
            1a 26180bcb                       # unsigned(639110091)
            71                                # text(17)
               30303a30303a31303a35343a33323a3130 # "00:00:10:54:32:10"
            1a 3d6bbe90                       # unsigned(1030471312)
            67                                # text(7)
               32333239383336                 # "2329836"
            1a 35ecbb3d                       # unsigned(904706877)
            67                                # text(7)
               64796e616d6963                 # "dynamic"
            1a 13038bb5                       # unsigned(318999477)
            67                                # text(7)
               756e6b6e6f776e                 # "unknown"
            1a 09e1fa37                       # unsigned(165804599)
            66                                # text(6)
               616374697665                   # "active"
```

The following figure shows the example Patch payload:

```
-------------------------JSON example ----------------------
 {
     {
      0x200a297b : "author1",
      0x35e1c1da : "book2"}:
      { null : null},
     {
      0x200a297b : "author5"} :
       {0x26c3ca7e : 4444},
     {
      0x200a297b : "newauthor",
      0x35e1c1da : "newbook"}:
     { 0x0119ddec : 1,
       0x26c3ca7e : 1},

  0x5cc7979 : "favoured",
  0x2935c912 : null,
  0x3f83c748 : [ "example"],
  0x324d9526 : "+01-123-456-7890"
}
----------------------------CBOR code ----------------------
a7                                    # map(7)
   a2                                 # map(2)
      1a 200a297b                     # unsigned(537536891)
```

```
      67                                  # text(7)
         617574686f7231                   # "author1"
      1a 35e1c1da                         # unsigned(903987674)
      65                                  # text(5)
         626f6f6b32                       # "book2"
   a1                                     # map(1)
      00                                  # unsigned(0)
      f6                                  # primitive(22)
   a1                                     # map(1)
      1a 200a297b                         # unsigned(537536891)
      67                                  # text(7)
         617574686f7235                   # "author5"
   a1                                     # map(1)
      1a 26c3ca7e                         # unsigned(650365566)
      19 115c                             # unsigned(4444)
   a2                                     # map(2)
      1a 200a297b                         # unsigned(537536891)
      69                                  # text(9)
         6e6577617574686f72               # "newauthor"
      1a 35e1c1da                         # unsigned(903987674)
      67                                  # text(7)
         6e6577626f6f6b                   # "newbook"
   a2                                     # map(2)
      1a 0119ddec                         # unsigned(18472428)
      01                                  # unsigned(1)
      1a 26c3ca7e                         # unsigned(650365566)
      01                                  # unsigned(1)
   1a 05cc7979                            # unsigned(97286521)
   68                                     # text(8)
      6661766f75726564                    # "favoured"
   1a 2935c912                            # unsigned(691390738)
   f6                                     # primitive(22)
   1a 3f83c748                            # unsigned(1065600840)
   81                                     # array(1)
      67                                  # text(7)
         6578616d706c65                   # "example"
   1a 324d9526                            # unsigned(843945254)
   70                                     # text(16)
      2b30312d3132332d3435362d37383930    # "+01-123-456-7890"
```

Appendix D. Comparison with LWM2M

CoMI and LWM2M, both, provide RESTful device management services over CoAP. Differences between the designs are highlighted in this section.

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 subresources. 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.

Appendix E. 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 2: Probability of one or more clashes

In CoMI the YANG node names are hashed with the murmur3 hash algorithm. The consequence is that a clash may occur, and the YANG node name needs to be rehashed with a given prefix character. 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^bits is given by:

((N-1)/N)*((N-2)/N)*....(N-(k-1))/N

which can be approximated with:

exp(-k(k-1)/2N)

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

1 - exp(-k(k-1)/2N) ~ k(k-1)/2N

Table 2 shows the probabilities for a given set of values of N=2^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 that one pair of hashes clashes out of k hashes is given by:

k*(k-1)/2N

The probability that the remaining k-2 hashes do no clash is given by:

exp(-(k-3)(k-2)/2N)

The probability that exactly one pair of hashes clashes is given by:

(k*(k-1)/2N) * exp(-(k-3)(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:

k(k-1)/2N - (k*(k-1)/2N) * exp(-(k-3)(k-2)/2N) =

(k(k-1)/2N) (1-exp(-(k-3)(k-2)/2N)) =

(k(k-1)(k-3)(k-2)/4N^2)

Number of names	28 bits	29 bits	30 bits	31 bits	32 bits	33 bits
10	1,8E-14	4,4E-15	1,1E-15	2,7E-16	6,8E-17	1,7E-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-1	2,1E-11	5,3E-12
10^3	3,4E-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	3,4E-05

Table 3: Probability of more than 2 entries equal clashes

The corresponding probabilities are shown in Table 3. The probabilities of Table 3 include the probability that more than 2 hashes share one clashing value. Assuming a hash size of 2^30, and about 1000 YANG nodes in a server, the probability of one clashing

pair is 0.5*10^-3, and the probability that more clashes occur is 2*10^-7.

Appendix F. 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 E 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.

F.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)

F.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).

F.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:

- 500 data nodes per server
- 10 server types
- 30 modules
- Module name is on average 20 bytes
- 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*2*2 represents 40 bytes for 2 object names in 2 server types.

* Module name array: 30*20 represents 600 bytes.

* Server identifier table: 10*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.

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

Juergen Schoenwaelder
Jacobs University
Campus Ring 1
Bremen 28759
Germany

Email: j.schoenwaelder@jacobs-university.de

Anuj Sehgal
consultant
Campus Ring 1
Bremen 28759
Germany

Email: anuj@iurs.org

core
Internet-Draft
Intended status: Informational
Expires: April 7, 2016

P. van der Stok
A. Sehgal
Consultant

October 5, 2015

Patch Method for Constrained Application Protocol (CoAP)
draft-vanderstok-core-patch-02

Abstract

The existing Constrained Application Protocol (CoAP) PUT method only allows a complete replacement of a resource. This does not permit applications to perform partial resource modifications. In case of resources with larger or complex data, or in situations where a resource continuity is required, replacing a resource is not an option. Several applications using CoAP will need to perform partial resource modifications. This proposal adds new CoAP methods, PATCH and iPATCH, to modify an existing CoAP resource partially.

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 7, 2016.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
1.2. Terminology and Acronyms 3
2. PATCH (iPATCH) Method 3
2.1. A Simple PATCH (iPATCH) Example 4
2.2. Response Codes 6
2.3. Option Numbers 6
3. Error Handling 6
4. Security Considerations 7
5. IANA Considerations 8
6. Acknowledgements 8
7. Change log 8
8. References 9
8.1. Normative References 9
8.2. Informative References 10
Authors' Addresses 10

1. Introduction

This specification defines the new Constrained Application Protocol (CoAP) [RFC7252] methods, PATCH and iPATCH, which are used to apply partial modifications to a resource.

PATCH is also specified for HTTP in [RFC5789]. Most of the motivation for PATCH described in [RFC5789] also applies here. iPATCH is the idem-potent 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.

1.1. 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 [RFC2119].

1.2. Terminology and Acronyms

This document uses terminology defined in [RFC5789] and [RFC7252].

2. PATCH (iPATCH) Method

The PATCH (iPATCH) method requests 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 (iPATCH) 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].

PATCH is not safe and not idempotent conformant to HTTP PATCH specified in [RFC5789].

iPATCH is not safe but idempotent conformant to CoAP PUT specified in [RFC7252], Section 5.8.3.

An iPATCH request is idempotent to prevent bad outcomes from collisions between two iPATCH requests on the same resource in a similar time frame. These collisions can be detected with the MessageId and the source end-point provided by the CoAP protocol (see section 4.5 of [RFC7252].

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. When the PATCH request is over-specified (i.e. Request specifies modifications to attributes which do not exist in the server), The server MAY execute all modifications to existing attributes and return a response code 2.02 Accepted.

The atomicity applies to a single server. When a PATCH (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 (iPATCH) response can invalidate a cache conformant with the PUT response. Caching behaviour as function of the valid 2.xx response codes for PATCH (iPATCH) are:

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.

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 (iPATCH). Servers are required to support a subset of the content formats as specified in sections 12.3 and 5.10.3 of [RFC7252]. Servers MUST ensure that a received PATCH payload is appropriate for the type of resource identified by the target resource of the request.

Clients MUST choose to use PATCH (iPATCH) rather than PUT when the request affects partial updates of a given resource.

2.1. A Simple PATCH (iPATCH) Example

The example is taken over from [RFC6902], which specifies a JSON notation for PATCH operations. A resource located at www.example.com/object contains a target JSON document.

```
JSON document original state
            {
              "x-coord": 256,
              "y-coord": 45
            }

 REQ:
     PATCH CoAP://www.example.com/object
Content-Type: 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
             }
```

This example illustrates use of a hypothetical PATCH on the /object/x-coord of the existing resource "object". The 2.04 (Changed) response code is conforms with the CoAP PUT method.

The same example using the Content-Type application/merge-patch+json from [RFC7396] looks like:

```
JSON document original state
            {
              "x-coord": 256,
              "y-coord": 45
            }

 REQ:
     PATCH CoAP://www.example.com/object
Content-Type: application/merge-patch+json
     { "x-coord":45}
RET:
     CoAP 2.04 Changed

JSON document final state
            {
               "x-coord": 45,
               "y-coord": 45
             }
```

2.2. Response Codes

PATCH (iPATCH) for CoAP adopt the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252].

2.3. Option Numbers

PATCH for CoAP adopts the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

3. Error Handling

A PATCH (iPATCH) request may fail under certain known conditions. These situations should be dealt with as expressed below.

Malformed PATCH (iPATCH) payload: If a server determines that the payload provided with a PATCH (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 (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,

* the resource specified in the request becomes invalid by applying the payload -- 4.06 (Not Acceptable) CoAP Response Code,

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 (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. Security Considerations

This section analyses the possible threats to the CoAP PATCH (iPATCH) protocol. It is meant to inform protocol and application developers about the security limitations of CoAP PATCH (iPATCH) as described in this document. The security consideration of section 15 of [RFC2616], section 11 of [RFC7252], and section 5 of [RFC5789] also apply.

The security considerations for PATCH (iPATCH) are nearly identical to the security considerations for PUT ([RFC7252]). The mechanisms used for PUT can be used for PATCH (iPATCH) as well.

PATCH (iPATCH) is secured following the CoAP recommendations as specified in section 9 of [RFC7252]. When more appropriate security techniques are standardized for CoAP, PATCH (iPATCH) can also be secured by those new techniques.

5. IANA Considerations

The entry with name PATCH in the sub-registry, "CoAP Method Codes", is 0.05. The entry with name iPATCH in the sub-registry, "CoAP Method Codes", is 0.06. The additions will follow the "IETF Review or IESG Approval" procedure as described in [RFC5226].

A new response code must be entered to the sub-registry "CoAP response codes" which apply to the methods PATCH and iPATCH.

Code 4.09 with Description "Conflict" and described in this specification.

The addition to this sub-registry will follow the "IETF Review or IESG Approval" as described in [RFC5226].

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are needed for the following media type formats: "application/json-patch+json" [RFC6902], and "application/merge-patch+json" [RFC7396].

6. Acknowledgements

Klaus Hartke has pointed out some essential differences between CoAP and HTTP. We are grateful for discussions with Christian Amsuss, Carsten Bormann, Paul Duffy, Kovatsch Matthias, Michael Verschoor, Thomas Watteyne, and Gengyu Wei.

7. Change log

When published as a RFC, this section needs to be removed.

Version 0 to version 1:

- Changed patch motivation text.
- Removed sub-resource concept.
- Updated cache handling.
- Extended example.
- Update of error handling.

Version 1 to version 2

- section 3 rephrased use of error 4.09
- added conflict handling failure
- added idempotent iPATCH method

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>.

[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>.

8.2. Informative References

[I-D.vanderstok-core-comi]
Stok, P., Bierman, A., Schoenwaelder, J., and A. Sehgal, "CoAP Management Interface", draft-vanderstok-core-comi-07 (work in progress), July 2015.

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Anuj Sehgal
Consultant

Email: anuj@iurs.org

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.

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 Size1, Size2, 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]:

- action
- data node
- data tree
- module
- notification

- RPC
- schema node
- schema tree
- submodule

This specification also makes use of the following terminology:

- 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.
- CoOL client: The originating endpoint of a request, and the destination endpoint of a response.
- CoOL server: The destination endpoint of a request, and the originating endpoint of a response.
- delta: Within a list, a delta represents the different 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.
- 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.
- datastore: Resource used to store and access information.
- 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.
- event stream: Resource used to access notifications generated by a CoOL server. Events are defined using the YANG notification statement.
- function set: A group of well-known resources that provides a particular service.

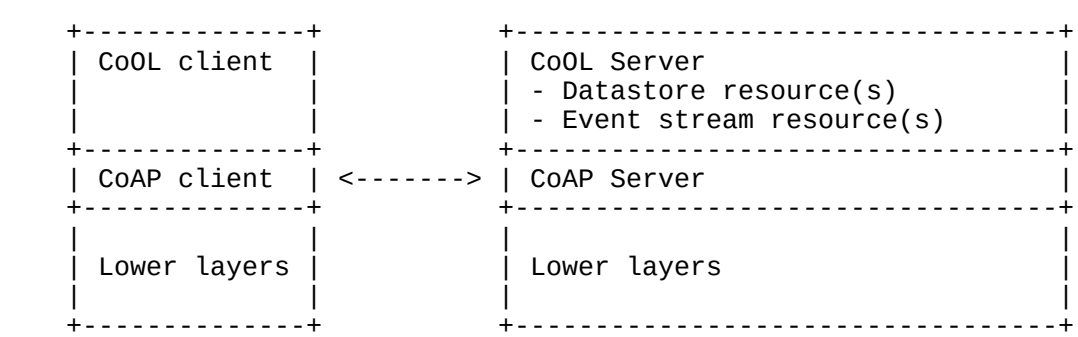
- object: Within CoOL, an object is a data node, an RPC or an action within a datastore resource or a notification within an event stream resource.
- parent: The collection in which a schema node is defined.
- resource: Content identified by a URI.
- running configuration datastore: A configuration datastore holding the complete configuration currently active on the device. The running configuration datastore always exists.
- 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.

- /c - The default datastore resource
- /c/c - The candidate configuration datastore resource
- /c/r - The running configuration datastore resource
- /c/b - The backup configuration datastore (use to implement rollbacks)
- /c/e - URI used to access the default event stream for this device.
- /c/e0, /c/e1, ... - URI used to access alternate event streams.
- /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:

- /c/1 is the default datastore resource for endpoint 1
- /c/1/c is the candidate datastore resource for endpoint 1
- /c/1/r is the running configuration datastore resource for endpoint 1
- /c/1/b is the backup configuration datastore resource for endpoint 1
- /c/1/e is the default event stream resource for endpoint 1
- /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:

- Immediate or differed commit of a candidate or backup datastore.
- Confirmed commit

- 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]:

- "/interfaces/interface" (SID 1533)
- "/interfaces/interface/description" (SID 1534)
- "/interfaces/interface/enabled" (SID 1535)
- "/interfaces/interface/name" (SID 1537)

- "/interfaces/interface/1538" (SID 1534)
- "/system-state/clock" (SID 1717)
- "/system-state/clock/boot-datetime" (SID 1718)
- "/system-state/clock/current-datetime" (SID 1719)
- "/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:

- If the data node requested is not implemented or not instantiated, the CBOR simple value 'undefined' is returned.
- 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.
- 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:

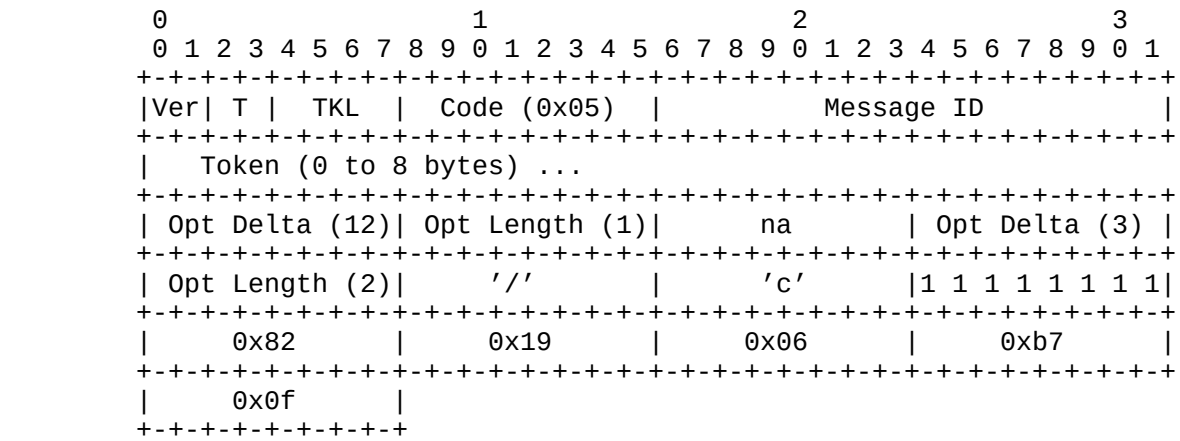
FETCH /c Content-Format(application/cool-instance-id-list+cbor)
[1719, +15]

CoAP response:

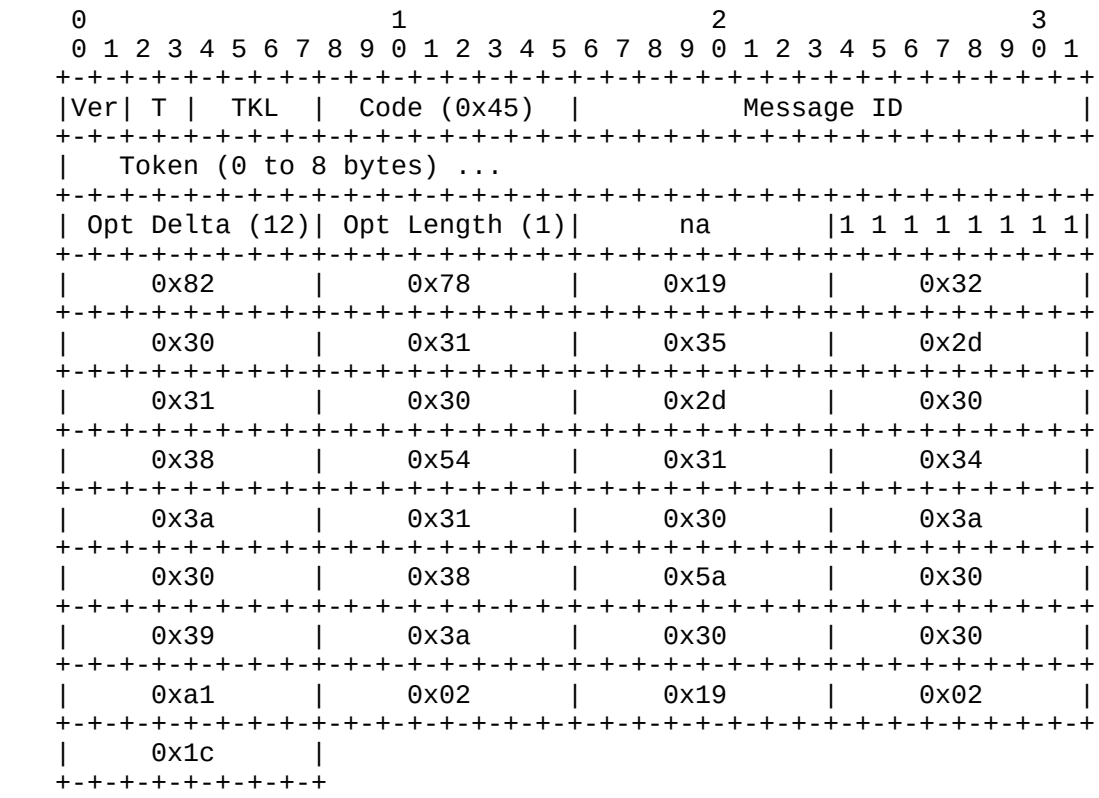
```
2.05 Content Content-Format(application/cool-value-list+cbor)
[
  "2015-10-08T14:10:08Z09:00",    # current-datetime (SID 1719)
  {                               # clock (SID 1734)
    +2 : 540                      # timezone-utc-offset (SID 1736)
  }
]
```

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:



CoAP response:



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 form 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:

- "/system/clock/timezone/timezone-utc-offset/timezone-utc-offset" (SID 1736)
- "/system/ntp/enabled" (SID 1751)
- "/system/ntp/server" (SID 1752)
- "/system/ntp/server/name" (SID 1755)
- "/system/ntp/server/prefer" (SID 1756)
- "/system/ntp/server/transport/udp/udp" (SID 1757)
- "/system/ntp/server/transport/udp/udp/address" (SID 1758)
- "/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:

- 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.

- If the targeted data instance doesn't exist, this instance is created.

- 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:

- Set "/system/ntp/enabled" (SID 1751) to true.

- Remove the server "tac.nrc.ca" from the"/system/ntp/server" (SID 1752) list.

- Add the server "NTP Pool server 2" to the list "/system/ntp/server" (SID 1752).

- 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 : {                            # udp (SID 1757)
        +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 sents 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)

- "/link-failure/if-name" (SID 1943)

- "/link-failure/admin-status" (SID 1944)

- "/interfaces/interface/interface-enabled" (SID 1538)

- "/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:

- a CBOR simple type 'default' is returned for each data node currently set to its default value
- a CBOR simple type 'undefined value' is returned for each data node not instantiated or not supported
- 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:

- The management application often needs a single, definitive, and complete set of configuration values that determine how the networking device works.
- Documentation about default values can be unreliable or unavailable.
- Some management applications might not have the capabilities to correctly parse and interpret formal data models.
- 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:

- Media type = application/cool-instance-id-list
- Encoding = CBOR
- ID = 61
- Reference = RFC XXXX

Second entry:

- Media type = application/cool-value
- Encoding = CBOR
- ID = 62
- Reference = RFC XXXX

Third entry:

- Media type = application/cool-value-list
- Encoding = CBOR
- ID = 63
- Reference = RFC XXXX

Fourth entry:

- Media type = application/cool-value-pairs+cbor
- Encoding = CBOR
- ID = 64
- 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.

- Value = 19
- Semantics = Default value
- 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 paylaod.";
    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 paylaod.";
    description
      "Definition of the payload of a notification transfered 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

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.

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:

- Reading Node: any regular node that reads information from the Sleepy Node.

- 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.

- Discovering Node: any regular node that performs discovery of the nodes in a network, including Sleepy Nodes.

- Destination Node: any regular node or node in a group that receives a message that is generated by the Sleepy Node.

- 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.

- 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.

- Delegated resource is the copy at the Proxy of a resource present in the Sleepy Node.

2.1. Node interactions and use cases

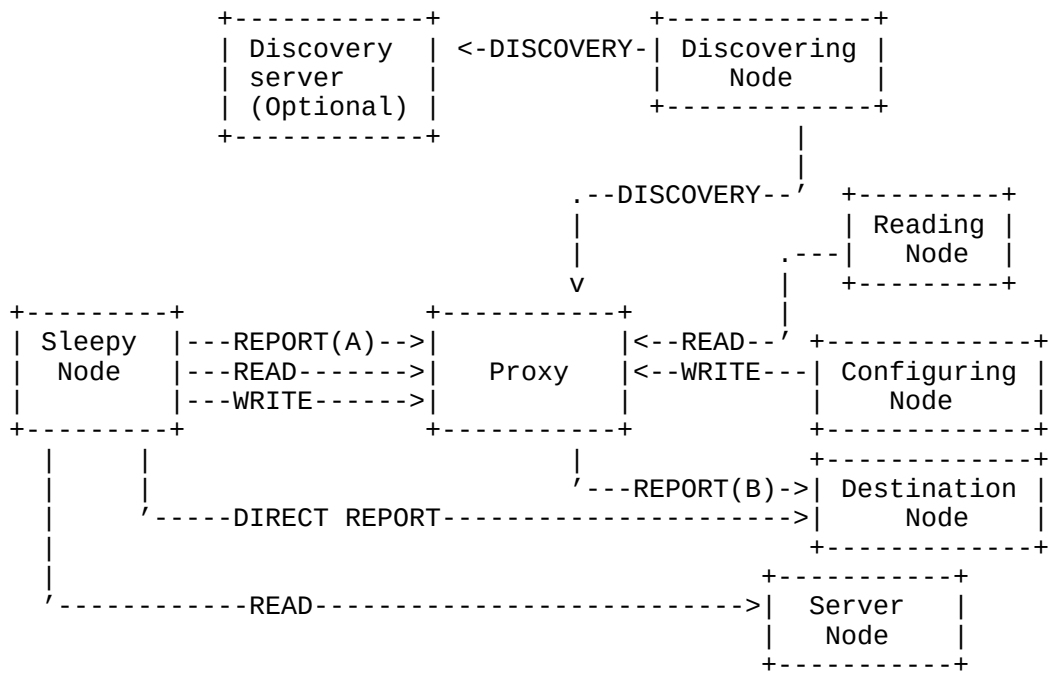


Figure 1: Interaction model for Sleepy Nodes in IP-based networks

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.

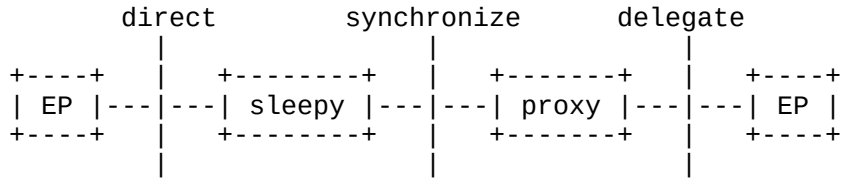


Figure 2: Architecture of Sleepy Node support

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.

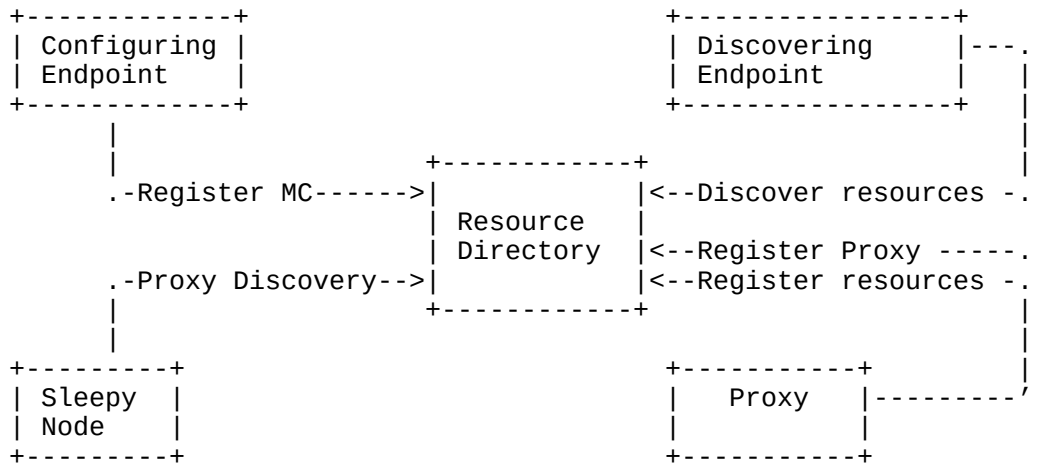


Figure 3: Interactions involving Resource Directory

5. Synchronize interface

The functions of the synchronize interface implemented by the Proxy are described in this section.

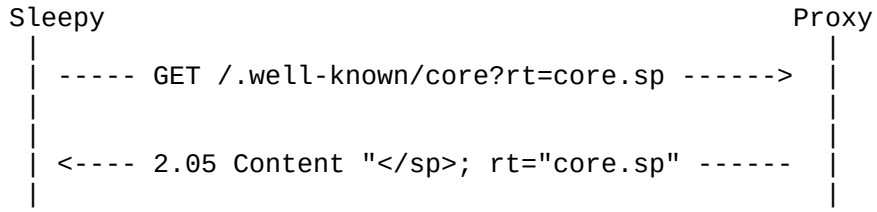
5.1. Sleepy Node discovers proxy

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.



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.

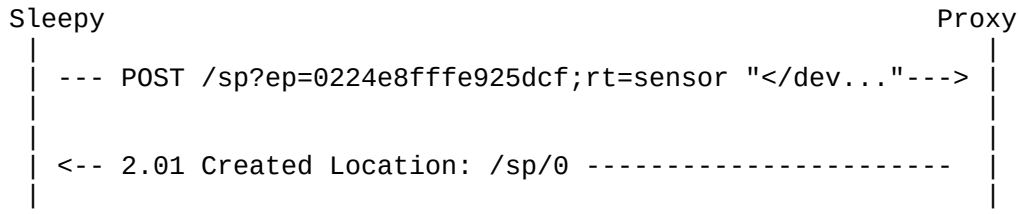
In case a Sleepy Node delegates its own resources to more than one Proxy and each Proxy registers the Sleepy Node’s resource in a RD, the RD entries from the different Proxies for the same Sleepy Node risk to overlap.

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.




```
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
```

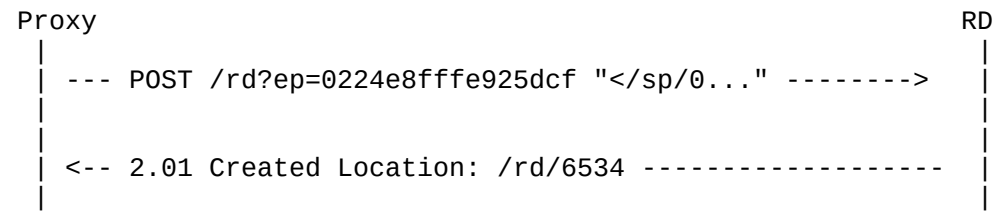
The delegated resource has been created with path /sp/0 on the Proxy in the example above. The path to the ep can be discovered as shown below:

```
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.



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.

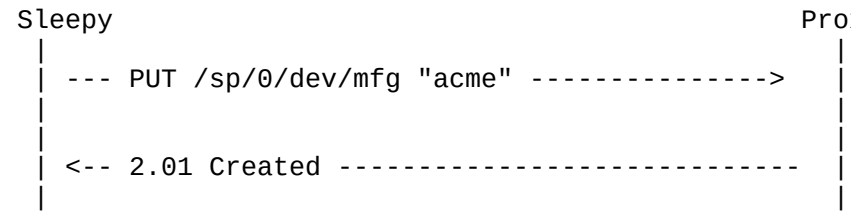
Success: 2.04 "Changed", the request MUST include the new representation of the delegated resource.

Success: 2.05 "Content", the response MUST include the current representation of the delegated resource.

Failure: 4.00 "Bad Request". Malformed request.

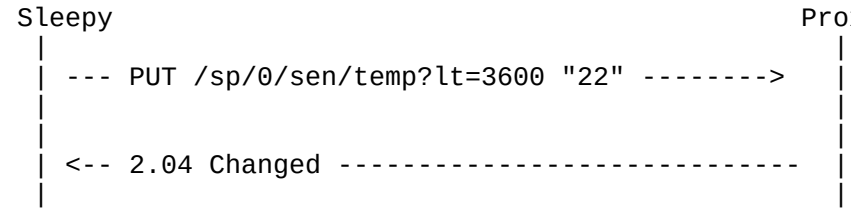
Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example describes how a Sleepy Node can initialize the resource containing its manufacturer name just after registration.



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.



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.

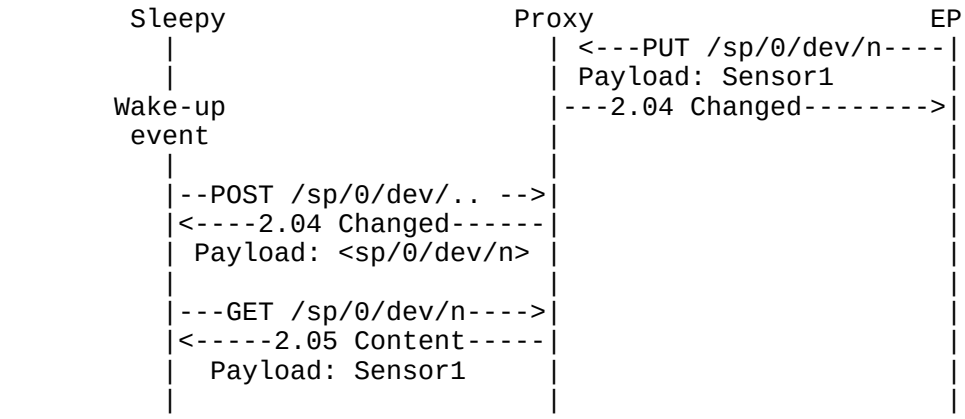


Figure 4: Example: A Sleepy Node READs resource updates from his Proxy

6. Delegate Interface

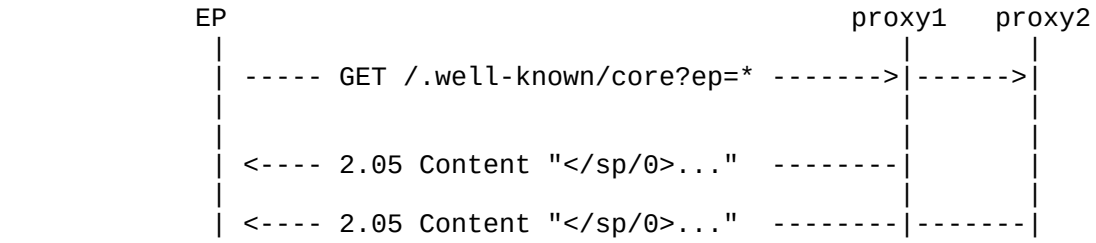
This section details the functions belonging to the delegate interface.

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].

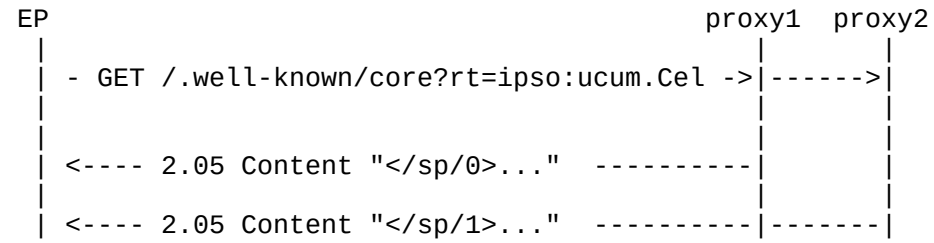
Resource discovery between an Endpoint and a proxy or an Endpoint and a RD needs special care to take into account the fact that resources from a Sleepy Node might appear duplicated. EPs SHOULD employ 2-step resource discovery by looking up Sleepy Nodes AND resource types to detect duplicate resources. 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.




```
Req: GET coap://[ff02::1]/.well-known/core?ep=*
Res: 2.05 Content
</sp/0>;ep="0224e8fffe925dcf"
Res: 2.05 Content
</sp/0>;ep="02004cfffe4f4f50"
</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.



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}

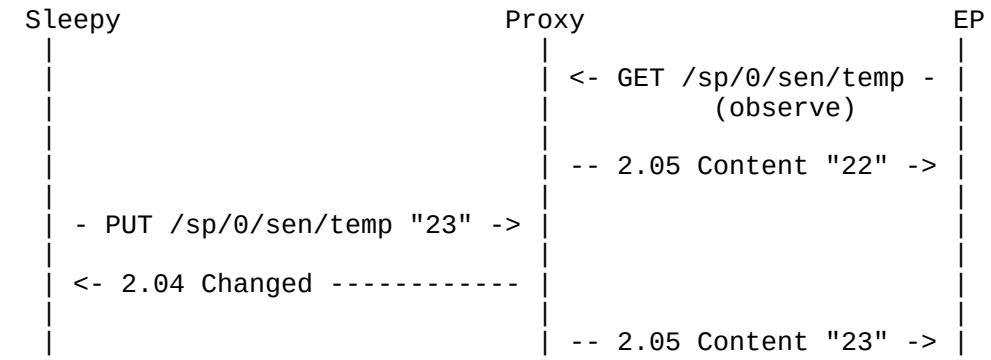
URI Template Variables:

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

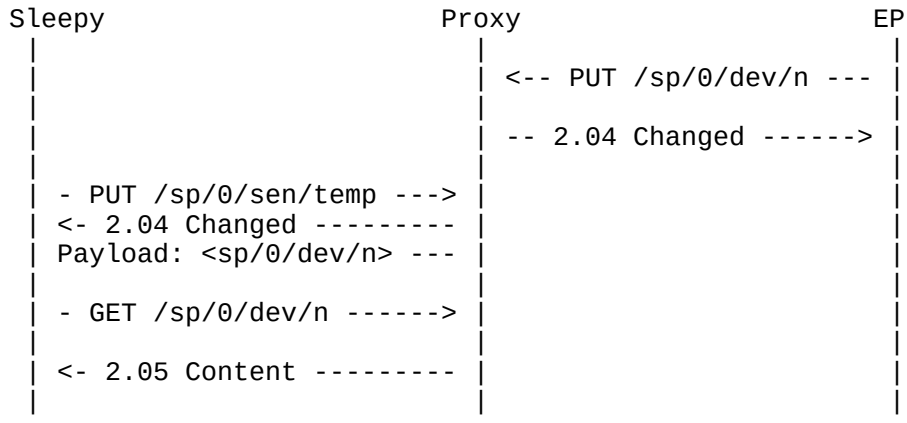
In the example below an EP observes the changes of temperature through the Proxy.



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.



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