Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2015

Resource Directory Names for Certificate Mode DTLS
draft-fossati-core-certmode-rd-names-00

Abstract

This memo describes the use of Resource Directory names in CoAP Certificate Mode DTLS for the purpose of verifying the identity of a server by a client endpoint.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

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

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

Today, many Internet of Things (IoT) deployments consist of an IoT device that interacts with a cloud service infrastructure. (This deployment model is described in Section 2.2 of [I-D.iab-smart-object-architecture].)

If TLS/DTLS is used to mutually authenticate the device and the cloud server, then the guidance in [I-D.ietf-dice-profile] - which, in turn, takes [RFC7252] recommendations into account - should be followed.

In particular, according to Section 9.1.3.3 of [RFC7252], a client that receives a certificate from the server must check that the authority of the requested URI matches "at least one of the authorities of any CoAP URI found in a field of URI type in the SubjectAltName (SAN) set. If there is no SubjectAltName in the certificate, then the authority of the request URI must match the Common Name (CN) found in the certificate [...]."

According to Section 4.2.1.6 of [RFC5280] an URI that includes an authority - such as a ‘coaps’ URI - needs to include a fully qualified domain name (FQDN), or an IP literal as its host part.
(So, an IoT device that wants to talk to a CoAP server at coaps://example.com will expect to receive a certificate with a matching URI in either the content of the SAN extension or the CN.)

The combination of the two requirements above, together with text in Section 3 of [RFC6066] which only allows FQDN hostname of the server in the ServerName field, basically binds Certificate Mode DTLS to either DNS, or static host tables containing FQDN’s mappings, or some other system for lookup of registered names which is able to fully mimic the DNS naming scheme.

While DNS can be taken for granted in the Web, CoAP networks do not mandate its presence. In fact, there are IoT deployments where the server infrastructure is located in a home or residential environment in which IoT devices interact with the server solely in the local network (see also Section 2.1 of [I-D.iab-smart-object-architecture]).

Since static configuration is not generally a viable option, in order to cope with scenarios like the one described above there is a need to define some kind of stable, non-DNS, identifier that can be used for ‘coaps’ URIs in Certificate Mode DTLS as a fall-back in case DNS is not deployed, or not understood by CoAP endpoints.

1.1. Challenges

There seem to be at least four challenges that need to be solved to make sure that the IoT device is indeed talking to a server whose X.509 certificate identity can be compared with the requested CoAP URI:

1. What identifiers should be used in the certificate?

2. What identifier should be contained in the hostname part of the endpoint URI?

3. What identifier should be communicated in the SNI during the TLS/DTLS exchange?

4. How can the identifier in the CoAP URI be mapped to an IP address?

The way the Web solves these problems is by assuming that the name of an application service is based on a DNS domain name, as stated in [RFC6125]. The identifiers used in the certificate and in the SNI are then FQDN’s.
In order to offer a solution for the CoAP space this document suggests the use of Resource Directory endpoint names (and domains) as an alternative to DNS names.

2. Terminology and Requirements Language

This specification requires the reader to be familiar with the terminology used in documents produced by the CoRE, TLS, and PKIX working groups.

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

3. Resource Directory Names and Domains

In CoAP networks, a Resource Directory (RD) [I-D.ietf-core-resource-directory] is an entity that acts as a centralized store where protocol endpoints can register and lookup links to resources that are made available in the network. The RD defines the concept of an "endpoint name" which identifies a given Endpoint (i.e. web server) within a given "domain". Under the assumption of its uniqueness, an endpoint name/domain can be used as a stable host component for CoAP authorities.

3.1. Uniqueness Guarantee

An endpoint name is guaranteed to be unique within the associated domain. If the domain is elided during registration, the RD should assure its uniqueness within an implicit default domain.

3.2. Authority Format

3.2.1. Requirements

The syntax for RD name authorities has been designed to satisfy the following requirements:

REQ#1: full compatibility with URI reg-name syntax;

REQ#2: support identifiers from different and independently administered sources (e.g. those defined in OMA spec, EUI-64 [EUI-64], etc.);

REQ#3: allow for an optional "domain" under which a given name exists (for compatibility with current RD spec).
3.2.2. Syntax

The following ABNF reuses ‘port’ from [RFC3986]; ALPHA and DIGIT from [RFC5234].

```
RD-char = ALPHA / DIGIT / "-" / "_" / "~" / "!" / "$" / ";" / ";" / ";" / ";" / ";" / ";"
RD-ns = ALPHA *(ALPHA / DIGIT / ";") ; the name-space
RD-name = 1*RD-char
RD-domain = 1*63RD-char
RD-authority = [ RD-ns "+" ] RD-name [ "." RD-domain ] [ ":" port ]
```

Note that RD-char is the set of chars allowed in reg-name (REQ#1) from which the two following characters have been removed:

- the dot ("."), which is used to introduce the domain component (REQ#3);
- the plus ("+"), which is used to encode namespace information along with the name in an unambiguous way (REQ#2).

If RD-ns is present, then the length of RD-ns and RD-name MUST be less than 63 chars.

Percent encoding MUST NOT be used if not needed, i.e. it can be used only to encode non otherwise allowed chars.

3.2.3. Examples

- eui-64+01-23-45-67-89-ab-cd-ef
- imei+123456789012345
- imei+123456789012345:9876
- uuid+64d5ecfa-addc-4695-ac6e-36e8b18de4b9
- eui-64+01-23-45-67-89-ab-cd-ef.local:1234
- name.domain:1234

3.2.4. Uri-Host and Uri-Port Considerations

When RD-authority is used in a ‘coaps’ URI, its value is the same as the ServerName.name included (and successfully validated) by the client in the associated DTLS handshake (see Section 3.3).
Hence, there is no need to include explicit Uri-Host and Uri-Port Options in requests associated to the same security context [[CREF1: This updates Sections 6.4 and 6.5 of [RFC7252]]].

If any of Uri-Host or Uri-Port is included in the request, then its value MUST match the corresponding value set in the established security context.

3.3. SNI Name Type and Server Name Syntax

In order to encode RD authorities in a ServerNameList, the extension_data field of the server_name extension is expanded to allow a RDAuthority in a ServerName:

```plaintext
struct {
    NameType name_type;
    select (name_type) {
        case host_name: HostName;
        case rd_authority: RDAuthority;
    } name;
} ServerName;

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

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

RDAuthority, the data structure associated with the rd_authority NameType, is a variable-length vector that begins with a 16-bit length field indicating the length of the following RD authority. The RD authority is represented as a byte string using ASCII encoding. It MUST NOT contain any percent-encoded character other than for those characters not explicitly allowed by the grammar in Section 3.2.

3.4. New OID arc for CoAP

This OID designates the OID arc for CoAP-related OIDs assigned by future IETF action, including those introduced by the present document:

```plaintext
id-coap OBJECT IDENTIFIER ::= { id-pkix coap(TODO) }
```
3.5. OtherName type-id and value Syntax

A X.509 Server Certificate intended to be used for resources served by a RD authority MUST contain an otherName SAN identified using a type-id of ‘id-rdauthority-san’:

```plaintext
id-rdauthority-san OBJECT IDENTIFIER ::= { id-coap 2 }
```

The value field of the otherName MUST contain an RD authority (Section 3.2), encoded as a IA5String.

4. Client Behaviour

1) Send extended ClientHello containing:

   a) server_name extension with one (and one only) ServerName, case-insensitive matching the authority of the URI to be requested;

   b) Any other potentially useful extension, e.g. client_certificate_url;

2) Verify that the intended server name is indeed one of the identities bound to the presented certificate, by checking that the name in the SAN otherName of type id-rdauthority-san case-insensitive matches the authority requested via server_name;

3) Upon receiving the CertificateRequest message, send the certificate via a Certificate message - or CertificateURL message, if the client_certificate_url extension has been successfully negotiated during the "hello" phase;

4) Send ClientKeyExchange and then CertificateVerify to complete the mutual authentication process.

5. Server Behaviour

1) Server receives extended ClientHello carrying a server_name extension, and uses the given server_name (with a rd_authority NameType) to select the appropriate certificate. The selected certificate MUST include a SAN otherName with an id-rdauthority-san type-id and value, which MUST case-insensitive match the requested ServerName;

   a) If no certificate can be selected, the server MUST terminate the handshake by sending a fatal-level unrecognized_name(112) alert. [[CREF2: Prefer a single, hard failure, path over soft failure, or worse: ignoring the error altogether.]]
Rationale: do not waste time/energy; provide clear and prompt diagnostic to the peer. It doesn’t look like the condition that could be exploited by a timing attack.]

b) If a matching certificate exist, the server SHALL include an extension of type "server_name" in the (extended) ServerHello message with an empty value.

2) The server MUST send the selected certificate back to the client in the Certificate message.

3) Server MUST then request the client certificate via a CertificateRequest message and conclude its negotiation with a ServerHelloDone message.

4) When server receives the Certificate message from the client then, depending on the specific application security policy, it MAY want to match one of the identities of the client against a configured ACL, and decide whether to continue or to tear down the session [[CREF3: TODO Which alert code to use if ACL check fails?]].

5) The server application running on top of DTLS MUST check the requested URI authority case-insensitive matches the requested server_name.

6. IANA Considerations

[[CREF4: Need to register a few new IDs, not sure where (IANA, PKIX registry, TLS registry)?]]

- id-coap
- OtherName.type-id::id-rdauthority-san
- NameType::rd_authority
- ServerName.name::RDAuthority

7. Security Considerations

It’s the responsibility of the CA, by means of its Registration Authority component, to verify the identity of the requester before issuing a new certificate. In particular, the CA MUST ensure that no more than one certificate per SAN is valid at any given time. This should exclude the threat of a (possibly rogue) node to successfully impersonate another node’s identity.
Security considerations from Section 11.1 of [RFC6066] fully apply.

8. Acknowledgements

TODO

9. References

9.1. Normative References


[I-D.ietf-core-resource-directory]

[I-D.ietf-dice-profile]


9.2. Informative References

[I-D.iab-smart-object-architecture]
Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson,
"Architectural Considerations in Smart Object Networking",
draft-iab-smart-object-architecture-06 (work in progress),
October 2014.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and
Verification of Domain-Based Application Service Identity
within Internet Public Key Infrastructure Using X.509
(PKIX) Certificates in the Context of Transport Layer

Authors' Addresses

Thomas Fossati
Alcatel-Lucent
3 Ely Road
Milton, Cambridge  CB24 6DD
Great Britain

Email: thomas.fossati@alcatel-lucent.com

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

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

CoAP is a RESTful transfer protocol for constrained nodes and networks. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications will need to transfer larger payloads -- for instance, for firmware updates. With HTTP, TCP does the grunt work of slicing large payloads up into multiple packets and ensuring that they all arrive and are handled in the right order.

CoAP is based on datagram transports such as UDP or DTLS, which limits the maximum size of resource representations that can be transferred without too much fragmentation. Although UDP supports larger payloads through IP fragmentation, it is limited to 64 KiB and, more importantly, doesn’t really work well for constrained applications and networks.

Instead of relying on IP fragmentation, this specification extends basic CoAP with a pair of "Block" options, for transferring multiple blocks of information from a resource representation in multiple request-response pairs. In many important cases, the Block options enable a server to be truly stateless: the server can handle each block transfer separately, with no need for a connection setup or other server-side memory of previous block transfers.

In summary, the Block options provide a minimal way to transfer larger representations in a block-wise fashion.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 30, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1.  Introduction ................................................. 3
2.  Block-wise transfers ......................................... 5
   2.1. The Block2 and Block1 Options .......................... 5
   2.2. Structure of a Block Option ............................. 6
   2.3. Block Options in Requests and Responses ............... 8
   2.4. Using the Block2 Option ................................ 10
   2.5. Using the Block1 Option ................................ 12
   2.6. Combining Blockwise Transfers with the Observe Option .. 13
   2.7. Combining Block1 and Block2 ............................. 14
   2.8. Combining Block2 with Multicast ........................ 14
   2.9. Response Codes ......................................... 14
      2.9.1. 2.31 Continue .................................... 15
      2.9.2. 4.08 Request Entity Incomplete ..................... 15
      2.9.3. 4.13 Request Entity Too Large ...................... 15
   2.10. Caching Considerations ................................ 15
3.  Examples .................................................... 16
   3.1. Block2 Examples ........................................ 16
   3.2. Block1 Examples ........................................ 20
   3.3. Combining Block1 and Block2 ............................. 21
   3.4. Combining Observe and Block2 ............................ 23
4.  The Size2 and Size1 Options ................................ 26
5.  HTTP Mapping Considerations ................................ 27
6.  IANA Considerations ......................................... 29
7.  Security Considerations ..................................... 29
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.)

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.

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 identifying these options, we use the number 1 to refer to the transfer of the resource representation that pertains to the request, and the number 2 to refer to the transfer of the resource representation for the response.

In the following, the term "payload" will be used for the actual content of a single CoAP message, i.e. a single block being transferred, while the term "body" will be used for the entire resource representation that is being transferred in a block-wise fashion. The Content-Format option applies to the body, not to the payload, in particular the boundaries between the blocks may be in places that are not separating whole units in terms of the structure, encoding, or content-coding used by the Content-Format.

In most cases, all blocks being transferred for a body (except for the last one) will be of the same size. The block size is not fixed by the protocol. To keep the implementation as simple as possible, the Block options support only a small range of power-of-two block sizes, from \(2^4\) (16) to \(2^{10}\) (1024) bytes. As bodies often will not evenly divide into the power-of-two block size chosen, the size need not be reached in the final block (but even for the final block, the chosen power-of-two size will still be indicated in the block size field of the Block option).

2.1. The Block2 and Block1 Options
Both Block1 and Block2 options can be present both in request and response messages. In either case, the Block1 Option pertains to the request payload, and the Block2 Option pertains to the response payload.

Hence, for the methods defined in [RFC7252], Block1 is useful with the payload-bearing POST and PUT requests and their responses. Block2 is useful with GET, POST, and PUT requests and their payload-bearing responses (2.01, 2.02, 2.04, 2.05 -- see section "Payload" of [RFC7252]).

Where Block1 is present in a request or Block2 in a response (i.e., in that message to the payload of which it pertains) it indicates a block-wise transfer and describes how this specific block-wise payload forms part of the entire body being transferred ("descriptive usage"). Where it is present in the opposite direction, it provides additional control on how that payload will be formed or was processed ("control usage").

Implementation of either Block option is intended to be optional. However, when it is present in a CoAP message, it MUST be processed (or the message rejected); therefore it is identified as a critical option. It MUST NOT occur more than once.

2.2. Structure of a Block Option

Three items of information may need to be transferred in a Block (Block1 or Block2) option:

- The size of the block (SZX);
- whether more blocks are following (M);
- 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](image)

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.

* Finally, the SZX block size given in a control Block1 Option indicates the largest block size preferred by the server for transfers toward the resource that is the same or smaller than the one used in the initial exchange; the client SHOULD use this block size or a smaller one in all further requests in the transfer sequence, even if that means changing the block size (and possibly scaling the block number accordingly) from now on.

Using one or both Block options, a single REST operation can be split into multiple CoAP message exchanges. As specified in [RFC7252], each of these message exchanges uses their own CoAP Message ID.

The Content-Format Option sent with the requests or responses MUST reflect the content-format of the entire body. If blocks of a response body arrive with different content-format options, it is up to the client how to handle this error (it will typically abort any ongoing block-wise transfer). If blocks of a request arrive at a server with mismatching content-format options, the server MUST NOT assemble them into a single request; this usually leads to a 4.08 (Request Entity Incomplete, Section 2.9.2) error response on the mismatching block.

2.4. Using the Block2 Option

When a request is answered with a response carrying a Block2 Option with the M bit set, the requester may retrieve additional blocks of the resource representation by sending further requests with the same options as the initial request and a Block2 Option giving the block number and block size desired. In a request, the client MUST set the M bit of a Block2 Option to zero and the server MUST ignore it on reception.

To influence the block size used in a response, the requester MAY also use the Block2 Option on the initial request, giving the desired size, a block number of zero and an M bit of zero. A server MUST use the block size indicated or a smaller size. Any further block-wise requests for blocks beyond the first one MUST indicate the same block size that was used by the server in the response for the first request that gave a desired size using a Block2 Option.

Once the Block2 Option is used by the requester and a first response has been received with a possibly adjusted block size, all further requests in a single block-wise transfer SHOULD ultimately use the same size, except that there may not be enough content to fill the
last block (the one returned with the M bit not set). (Note that the client may start using the Block2 Option in a second request after a first request without a Block2 Option resulted in a Block2 option in the response.) The server SHOULD use the block size indicated in the request option or a smaller size, but the requester MUST take note of the actual block size used in the response it receives to its initial request and proceed to use it in subsequent requests. The server behavior MUST ensure that this client behavior results in the same block size for all responses in a sequence (except for the last one with the M bit not set, and possibly the first one if the initial request did not contain a Block2 Option).

Block-wise transfers can be used to GET resources the representations of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing resources. In the latter case, the Block2 Option SHOULD be used in conjunction with the ETag Option, to ensure that the blocks being reassembled are from the same version of the representation: The server SHOULD include an ETag option in each response. If an ETag option is available, the client’s reassembler, when reassembling the representation from the blocks being exchanged, MUST compare ETag Options. If the ETag Options do not match in a GET transfer, the requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server MAY cache the current value of a representation for an ongoing sequence of requests. (The server may identify the sequence by the combination of the requesting end-point and the URI being the same in each block-wise request.) Note well that this specification makes no requirement for the server to establish any state; however, servers that offer quickly changing resources may thereby make it impossible for a client to ever retrieve a consistent set of blocks. Clients that want to retrieve all blocks of a resource SHOULD strive to do so without undue delay. Servers can fully expect to be free to discard any cached state after a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) after the last access to the state, however, there is no requirement to always keep the state for as long.

The Block2 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise response payload transfer (e.g., GET) operations to the same resource. This is rarely a requirement, but as a workaround, a client may vary the cache key (e.g., by using one of several URIs accessing resources with the same semantics, or by varying a proxy-safe elective option).
2.5. Using the Block1 Option

In a request with a request payload (e.g., PUT or POST), the Block1 Option refers to the payload in the request (descriptive usage).

In response to a request with a payload (e.g., a PUT or POST transfer), the block size given in the Block1 Option indicates the block size preference of the server for this resource (control usage). Obviously, at this point the first block has already been transferred by the client without benefit of this knowledge. Still, the client SHOULD heed the preference indicated and, for all further blocks, use the block size preferred by the server or a smaller one. Note that any reduction in the block size may mean that the second request starts with a block number larger than one, as the first request already transferred multiple blocks as counted in the smaller size.

To counter the effects of adaptation layer fragmentation on packet delivery probability, a client may want to give up retransmitting a request with a relatively large payload even before MAX_RETRANSMIT has been reached, and try restating the request as a block-wise transfer with a smaller payload. Note that this new attempt is then a new message-layer transaction and requires a new Message ID. (Because of the uncertainty whether the request or the acknowledgement was lost, this strategy is useful mostly for idempotent requests.)

In a blockwise transfer of a request payload (e.g., a PUT or POST) that is intended to be implemented in an atomic fashion at the server, the actual creation/replacement takes place at the time the final block, i.e. a block with the M bit unset in the Block1 Option, is received. In this case, all success responses to non-final blocks carry the response code 2.31 (Continue, Section 2.9.1). If not all previous blocks are available at the server at the time of processing the final block, the transfer fails and error code 4.08 (Request Entity Incomplete, Section 2.9.2) MUST be returned. A server MAY also return a 4.08 error code for any (final or non-final) Block1 transfer that is not in sequence; clients that do not have specific mechanisms to handle this case therefore SHOULD always start with block zero and send the following blocks in order.

One reason that a client might encounter a 4.08 error code is that the server has already timed out and discarded the partial request body being assembled. Clients SHOULD strive to send all blocks of a request without undue delay. Servers can fully expect to be free to discard any partial request body when a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) has elapsed after the most recent block
was transferred; however, there is no requirement on a server to always keep the partial request body for as long.

The error code 4.13 (Request Entity Too Large) can be returned at any time by a server that does not currently have the resources to store blocks for a block-wise request payload transfer that it would intend to implement in an atomic fashion. (Note that a 4.13 response to a request that does not employ Block1 is a hint for the client to try sending Block1, and a 4.13 response with a smaller SZX in its Block1 option than requested is a hint to try a smaller SZX.)

The Block1 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations to the same resource. Starting a new block-wise sequence of requests to the same resource (before an old sequence from the same endpoint was finished) simply overwrites the context the server may still be keeping. (This is probably exactly what one wants in this case - the client may simply have restarted and lost its knowledge of the previous sequence.)

2.6. Combining Blockwise Transfers with the Observe Option

The Observe Option provides a way for a client to be notified about changes over time of a resource [I-D.ietf-core-observe]. 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 blockwise 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 blockwise transfers, it SHOULD take note of the block size and apply it as a maximum size to all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the entry in that list is updated by the server receiving a new GET request for the resource from the client).

When sending a 2.05 (Content) notification, the server only sends the first block of the representation. The client retrieves the rest of the representation as if it had caused this first response by a GET request, i.e., by using additional GET requests with Block2 options containing NUM values greater than zero. (This results in the transfer of the entire representation, even if only some of the blocks have changed with respect to a previous notification.)
As with other dynamically changing resources, to ensure that the blocks being reassembled are from the same version of the representation, the server SHOULD include an ETag option in each response, and the reassembling client MUST compare the ETag options (Section 2.4). Even more so than for the general case of Block2, clients that want to retrieve all blocks of a resource they have been notified about with a first block SHOULD strive to do so without undue delay.

See Section 3.4 for examples.

2.7. Combining Block1 and Block2

In PUT and particularly in POST exchanges, both the request body and the response body may be large enough to require the use of block-wise transfers. First, the Block1 transfer of the request body proceeds as usual. In the exchange of the last slice of this block-wise transfer, the response carries the first slice of the Block2 transfer (NUM is zero). To continue this Block2 transfer, the client continues to send requests similar to the requests in the Block1 phase, but leaves out the Block1 options and includes a Block2 request option with non-zero NUM.

Block2 transfers that retrieve the response body for a request that used Block1 MUST be performed in sequential order.

2.8. Combining Block2 with Multicast

A client can use the Block2 option in a multicast GET request with NUM = 0 to aid in limiting the size of the response.

Similarly, a response to a multicast GET request can use a Block2 option with NUM = 0 if the representation is large, or to further limit the size of the response.

In both cases, the client retrieves any further blocks using unicast exchanges; in the unicast requests, the client SHOULD heed any block size preferences indicated by the server in the response to the multicast request.

Other uses of the Block options in conjunction with multicast messages are for further study.

2.9. Response Codes

Two response codes are defined by this specification beyond those already defined in [RFC7252], and another response code is extended in its meaning.
2.9.1. 2.31 Continue

This new success status code indicates that the transfer of this block of the request body was successful and that the server encourages sending further blocks, but that a final outcome of the whole block-wise request cannot yet be determined. No payload is returned with this response code.

2.9.2. 4.08 Request Entity Incomplete

This new client error status code indicates that the server has not received the blocks of the request body that it needs to proceed. The client has not sent all blocks, not sent them in the order required by the server, or has sent them long enough ago that the server has already discarded them.

2.9.3. 4.13 Request Entity Too Large

In [RFC7252], section 5.9.2.9, the response code 4.13 (Request Entity Too Large) is defined to be like HTTP 413 "Request Entity Too Large". [RFC7252] also recommends that this response SHOULD include a Size1 Option (Section 4) to indicate the maximum size of request entity the server is able and willing to handle, unless the server is not in a position to make this information available.

The present specification allows the server to return this response code at any time during a Block1 transfer to indicate that it does not currently have the resources to store blocks for a transfer that it would intend to implement in an atomic fashion. It also allows the server to return a 4.13 response to a request that does not employ Block1 as a hint for the client to try sending Block1. Finally, a 4.13 response to a request with a Block1 option (control usage, see Section 2.3) where the response carries a smaller SZX in its Block1 option is a hint to try that smaller SZX.

2.10. Caching Considerations

This specification attempts to leave a variety of implementation strategies open for caches, in particular those in caching proxies. E.g., a cache is free to cache blocks individually, but also could wait to obtain the complete representation before it serves parts of it. Partial caching may be more efficient in a cross-proxy (equivalent to a streaming HTTP proxy). A cached block (partial cached response) can be used in place of a complete response to satisfy a block-wise request that is presented to a cache. Note that different blocks can have different Max-Age values, as they are transferred at different times. A response with a block updates the freshness of the complete representation. Individual blocks can be
validated, and validating a single block validates the complete representation. A response with a Block1 Option in control usage with the M bit set invalidates cached responses for the target URI.

A cache or proxy that combines responses (e.g., to split blocks in a request or increase the block size in a response, or a cross-proxy) may need to combine 2.31 and 2.01/2.04 responses; a stateless server may be responding with 2.01 only on the first Block1 block transferred, which dominates any 2.04 responses for later blocks.

If-None-Match only works correctly on Block1 requests with (NUM=0) and MUST NOT be used on Block1 requests with NUM ! = 0.

3. Examples

This section gives a number of short examples with message flows for a block-wise GET, and for a PUT or POST. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and examples for the operation of the block size negotiation.

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

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 128 bytes of payload each, and third ACK contains between 1 and 128 bytes.
In the second example (Figure 3), the client anticipates the blockwise 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.

CLIENT

CON [MID=1234], GET, /status, 2:0/0/64  ------>
| <------  ACK [MID=1234], 2.05 Content, 2:0/1/64 |
| CON [MID=1235], GET, /status, 2:1/0/64  ------>
| <------  ACK [MID=1235], 2.05 Content, 2:1/1/64 |
| CON [MID=1236], GET, /status, 2:2/0/64  ------>
| <------  ACK [MID=1236], 2.05 Content, 2:2/0/64 |

Figure 3: Blockwise GET with early negotiation

In the third example (Figure 4), the client is surprised by the need for a blockwise 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: Blockwise 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: Blockwise GET with late negotiation and lost CON

Figure 6: Blockwise 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.

CLIENT | SERVER
---|---
CON [MID=1234], PUT, /options, 1:0/1/128   ------>   
<------  ACK [MID=1234], 2.31 Continue, 1:0/1/128

CON [MID=1235], PUT, /options, 1:1/1/128   ------>   
<------  ACK [MID=1235], 2.31 Continue, 1:1/1/128

CON [MID=1236], PUT, /options, 1:2/0/128   ------>   
<------  ACK [MID=1236], 2.04 Changed, 1:2/0/128

Figure 7: Simple atomic blockwise 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.
Finally, a server receiving a blockwise 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.

3.3. Combining Block1 and Block2

Block options may be used in both directions of a single exchange. The following example demonstrates a blockwise POST request, resulting in a separate blockwise response.
Figure 10: Atomic blockwise POST with blockwise response

This model does provide for early negotiation input to the Block2 blockwise transfer, as shown below.
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 blockwise 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 blockwise 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 blockwise transfers handle tokens like any other CoAP exchange. As usual the client is free to choose tokens for each exchange as it likes.)

In the following example, the client also uses early negotiation to limit the block size to 64 bytes.

```
CLIENT          SERVER
|-----| Header: GET 0x41011636
    | Token: 0xfb
    | Uri-Path: status-icon
    | Observe: (empty)
    | Block2: 0/0/64

<------| Header: 2.05 0x61451636
    | Token: 0xfb
    | Block2: 0/1/64
    | Observe: 62350
    | ETag: 6f00f38e
    | Max-Age: 60
    | Payload: [64 bytes]

    ... (Usual GET transfer left out)

|-----| Header: 2.05 0x6145af9c
    | Token: 0xfb
    | Block2: 0/1/64
    | Observe: 62354
    | ETag: 6f00f392
    | Payload: [64 bytes]

|-----| Header: 0x6000af9c

|-----| Header: GET 0x41011637
    | Token: 0xfc
    | Uri-Path: status-icon
    | Block2: 1/0/64

<------| Header: 2.05 0x61451637
    | Token: 0xfc
    | Block2: 1/1/64
    | ETag: 6f00f392
```

Bormann & Shelby Expires April 30, 2015 [Page 25]
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 is already defined in [RFC7252] for 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 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").
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.

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>Size1</td>
<td>uint</td>
<td>0-4</td>
<td>(none)</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>Size2</td>
<td>uint</td>
<td>0-4</td>
<td>(none)</td>
</tr>
</tbody>
</table>

Table 2: Size Option Numbers

Implementation Notes:

- As a quality of implementation consideration, blockwise 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 blockwise 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]:

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Block2</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>27</td>
<td>Block1</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>28</td>
<td>Size2</td>
<td>[RFCXXXX]</td>
</tr>
</tbody>
</table>

Table 3: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [RFC7252]:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.31</td>
<td>Continue</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>4.08</td>
<td>Request Entity Incomplete</td>
<td>[RFCXXXX]</td>
</tr>
</tbody>
</table>

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.

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 blockwise requests may induce the server to create state, e.g. to create a snapshot for the blockwise 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 blockwise transfers in an expedient way to minimize the likelihood of running into such a timeout.
7.2. Mitigating Amplification Attacks

[RFC7252] discusses the susceptibility of CoAP end-points for use in amplification attacks.

A CoAP server can reduce the amount of amplification it provides to an attacker by offering large resource representations only in relatively small blocks. With this, e.g., for a 1000-byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

8. Acknowledgements

Much of the content of this draft is the result of discussions with the [RFC7252] authors, and via many CoRE WG discussions.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version. Esko Dijk reviewed a more recent version, leading to a number of further editorial improvements, a solution to the 4.13 ambiguity problem, and the section about combining Block and multicast. Markus Becker proposed getting rid of an ill-conceived default value for the Block2 and Block1 options. Peter Bigot insisted on a more systematic coverage of the options and response code.

Kepeng Li, Linyi Tian, and Barry Leiba wrote up an early version of the Size Option, which has informed this draft. Klaus Hartke wrote some of the text describing the interaction of Block2 with Observe. Matthias Kovatsch provided a number of significant simplifications of the protocol.

9. References

9.1. Normative References

[I-D.ietf-core-observe]


9.2. Informative References


Authors' Addresses

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

Phone: +49-421-218-63921
Email: cabo@tzi.org
Block-wise transfers in CoAP
draft-ietf-core-block-17

Abstract

CoAP is a RESTful transfer protocol for constrained nodes and networks. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications will need to transfer larger payloads -- for instance, for firmware updates. With HTTP, TCP does the grunt work of slicing large payloads up into multiple packets and ensuring that they all arrive and are handled in the right order.

CoAP is based on datagram transports such as UDP or DTLS, which limits the maximum size of resource representations that can be transferred without too much fragmentation. Although UDP supports larger payloads through IP fragmentation, it is limited to 64 KiB and, more importantly, doesn’t really work well for constrained applications and networks.

Instead of relying on IP fragmentation, this specification extends basic CoAP with a pair of "Block" options, for transferring multiple blocks of information from a resource representation in multiple request-response pairs. In many important cases, the Block options enable a server to be truly stateless: the server can handle each block transfer separately, with no need for a connection setup or other server-side memory of previous block transfers.

In summary, the Block options provide a minimal way to transfer larger representations in a block-wise fashion.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

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

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

Table of Contents

1. Introduction .............................................. 3
2. Block-wise transfers ...................................... 5
   2.1. The Block2 and Block1 Options ...................... 5
   2.2. Structure of a Block Option ....................... 6
   2.3. Block Options in Requests and Responses .......... 8
   2.4. Using the Block2 Option ............................ 10
   2.5. Using the Block1 Option ............................ 12
   2.6. Combining Block-wise Transfers with the Observe Option . 13
   2.7. Combining Block1 and Block2 ....................... 14
   2.8. Combining Block2 with Multicast ................... 14
   2.9. Response Codes ..................................... 14
       2.9.1. 2.31 Continue ................................ 15
       2.9.2. 4.08 Request Entity Incomplete ................ 15
       2.9.3. 4.13 Request Entity Too Large ................ 15
   2.10. Caching Considerations ............................ 15
3. Examples .................................................. 16
   3.1. Block2 Examples .................................... 16
   3.2. Block1 Examples .................................... 20
   3.3. Combining Block1 and Block2 ....................... 21
   3.4. Combining Observe and Block2 ....................... 23
4. The Size2 and Size1 Options ............................. 26
5. HTTP Mapping Considerations ............................. 27
6. IANA Considerations ..................................... 29
7. Security Considerations ................................ 29
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.)

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.

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 identifying these options, we use the number 1 to refer to the transfer of the resource representation that pertains to the request, and the number 2 to refer to the transfer of the resource representation for the response.

In the following, the term "payload" will be used for the actual content of a single CoAP message, i.e. a single block being transferred, while the term "body" will be used for the entire resource representation that is being transferred in a block-wise fashion. The Content-Format option applies to the body, not to the payload, in particular the boundaries between the blocks may be in places that are not separating whole units in terms of the structure, encoding, or content-coding used by the Content-Format.

In most cases, all blocks being transferred for a body (except for the last one) will be of the same size. The block size is not fixed by the protocol. To keep the implementation as simple as possible, the Block options support only a small range of power-of-two block sizes, from $2^{10}$ (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
Both Block1 and Block2 options can be present both in request and response messages. In either case, the Block1 Option pertains to the request payload, and the Block2 Option pertains to the response payload.

Hence, for the methods defined in [RFC7252], Block1 is useful with the payload-bearing POST and PUT requests and their responses. Block2 is useful with GET, POST, and PUT requests and their payload-bearing responses (2.01, 2.02, 2.04, 2.05 -- see section "Payload" of [RFC7252]).

Where Block1 is present in a request or Block2 in a response (i.e., in that message to the payload of which it pertains) it indicates a block-wise transfer and describes how this specific block-wise payload forms part of the entire body being transferred ("descriptive usage"). Where it is present in the opposite direction, it provides additional control on how that payload will be formed or was processed ("control usage").

Implementation of either Block option is intended to be optional. However, when it is present in a CoAP message, it MUST be processed (or the message rejected); therefore it is identified as a critical option. It MUST NOT occur more than once.

2.2. Structure of a Block Option

Three items of information may need to be transferred in a Block (Block1 or Block2) option:

- The size of the block (SZX);
- whether more blocks are following (M);
- 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.)

0
0 1 2 3 4 5 6 7
+++++++
| NUM | M | SZX |
+++++++

0
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+++++++
| NUM | M | SZX |
+++++++

0
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+++++++
| NUM | M | SZX |
+++++++

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 set 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.)

  o A Block2 Option in control usage in a request (e.g., GET):

    * The NUM field in the Block2 Option gives the block number of the payload that is being requested to be returned in the response.

    * In this case, the M bit has no function and MUST be set to zero.

    * The block size given (SZX) suggests a block size (in the case of block number 0) or repeats the block size of previous blocks received (in the case of a non-zero block number).

  o A Block1 Option in control usage in a response (e.g., a 2.xx response for a PUT or POST request):

    * The NUM field of the Block1 Option indicates what block number is being acknowledged.

    * If the M bit was set in the request, the server can choose whether to act on each block separately, with no memory, or whether to handle the request for the entire body atomically, or any mix of the two.

      + If the M bit is also set in the response, it indicates that this response does not carry the final response code to the request, i.e. the server collects further blocks from the same endpoint and plans to implement the request atomically (e.g., acts only upon reception of the last block of payload). In this case, the response MUST NOT carry a Block2 option.

      + Conversely, if the M bit is unset even though it was set in the request, it indicates the block-wise request was enacted now specifically for this block, and the response carries the final response to this request (and to any previous ones with the M bit set in the response’s Block1 Option in this sequence of block-wise transfers); the client is still
expected to continue sending further blocks, the request method for which may or may not also be enacted per-block.

* Finally, the SZX block size given in a control Block1 Option indicates the largest block size preferred by the server for transfers toward the resource that is the same or smaller than the one used in the initial exchange; the client SHOULD use this block size or a smaller one in all further requests in the transfer sequence, even if that means changing the block size (and possibly scaling the block number accordingly) from now on.

Using one or both Block options, a single REST operation can be split into multiple CoAP message exchanges. As specified in [RFC7252], each of these message exchanges uses their own CoAP Message ID.

The Content-Format Option sent with the requests or responses MUST reflect the content-format of the entire body. If blocks of a response body arrive with different content-format options, it is up to the client how to handle this error (it will typically abort any ongoing block-wise transfer). If blocks of a request arrive at a server with mismatching content-format options, the server MUST NOT assemble them into a single request; this usually leads to a 4.08 (Request Entity Incomplete, Section 2.9.2) error response on the mismatching block.

2.4. Using the Block2 Option

When a request is answered with a response carrying a Block2 Option with the M bit set, the requester may retrieve additional blocks of the resource representation by sending further requests with the same options as the initial request and a Block2 Option giving the block number and block size desired. In a request, the client MUST set the M bit of a Block2 Option to zero and the server MUST ignore it on reception.

To influence the block size used in a response, the requester MAY also use the Block2 Option on the initial request, giving the desired size, a block number of zero and an M bit of zero. A server MUST use the block size indicated or a smaller size. Any further block-wise requests for blocks beyond the first one MUST indicate the same block size that was used by the server in the response for the first request that gave a desired size using a Block2 Option.

Once the Block2 Option is used by the requester and a first response has been received with a possibly adjusted block size, all further requests in a single block-wise transfer SHOULD ultimately use the same size, except that there may not be enough content to fill the
last block (the one returned with the M bit not set). (Note that the client may start using the Block2 Option in a second request after a first request without a Block2 Option resulted in a Block2 option in the response.) The server SHOULD use the block size indicated in the request option or a smaller size, but the requester MUST take note of the actual block size used in the response it receives to its initial request and proceed to use it in subsequent requests. The server behavior MUST ensure that this client behavior results in the same block size for all responses in a sequence (except for the last one with the M bit not set, and possibly the first one if the initial request did not contain a Block2 Option).

Block-wise transfers can be used to GET resources the representations of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing resources. In the latter case, the Block2 Option SHOULD be used in conjunction with the ETag Option, to ensure that the blocks being reassembled are from the same version of the representation: The server SHOULD include an ETag option in each response. If an ETag option is available, the client's reassembler, when reassembling the representation from the blocks being exchanged, MUST compare ETag Options. If the ETag Options do not match in a GET transfer, the requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server MAY cache the current value of a representation for an ongoing sequence of requests. (The server may identify the sequence by the combination of the requesting end-point and the URI being the same in each block-wise request.) Note well that this specification makes no requirement for the server to establish any state; however, servers that offer quickly changing resources may thereby make it impossible for a client to ever retrieve a consistent set of blocks. Clients that want to retrieve all blocks of a resource SHOULD strive to do so without undue delay. Servers can fully expect to be free to discard any cached state after a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) after the last access to the state, however, there is no requirement to always keep the state for as long.

The Block2 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise response payload transfer (e.g., GET) operations to the same resource. This is rarely a requirement, but as a workaround, a client may vary the cache key (e.g., by using one of several URIs accessing resources with the same semantics, or by varying a proxy-safe elective option).
2.5. Using the Block1 Option

In a request with a request payload (e.g., PUT or POST), the Block1 Option refers to the payload in the request (descriptive usage).

In response to a request with a payload (e.g., a PUT or POST transfer), the block size given in the Block1 Option indicates the block size preference of the server for this resource (control usage). Obviously, at this point the first block has already been transferred by the client without benefit of this knowledge. Still, the client SHOULD heed the preference indicated and, for all further blocks, use the block size preferred by the server or a smaller one. Note that any reduction in the block size may mean that the second request starts with a block number larger than one, as the first request already transferred multiple blocks as counted in the smaller size.

To counter the effects of adaptation layer fragmentation on packet delivery probability, a client may want to give up retransmitting a request with a relatively large payload even before MAX_RETRANSMIT has been reached, and try restating the request as a block-wise transfer with a smaller payload. Note that this new attempt is then a new message-layer transaction and requires a new Message ID. (Because of the uncertainty whether the request or the acknowledgement was lost, this strategy is useful mostly for idempotent requests.)

In a block-wise transfer of a request payload (e.g., a PUT or POST) that is intended to be implemented in an atomic fashion at the server, the actual creation/replacement takes place at the time the final block, i.e. a block with the M bit unset in the Block1 Option, is received. In this case, all success responses to non-final blocks carry the response code 2.31 (Continue, Section 2.9.1). If not all previous blocks are available at the server at the time of processing the final block, the transfer fails and error code 4.08 (Request Entity Incomplete, Section 2.9.2) MUST be returned. A server MAY also return a 4.08 error code for any (final or non-final) Block1 transfer that is not in sequence; clients that do not have specific mechanisms to handle this case therefore SHOULD always start with block zero and send the following blocks in order.

One reason that a client might encounter a 4.08 error code is that the server has already timed out and discarded the partial request body being assembled. Clients SHOULD strive to send all blocks of a request without undue delay. Servers can fully expect to be free to discard any partial request body when a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) has elapsed after the most recent block.
was transferred; however, there is no requirement on a server to always keep the partial request body for as long.

The error code 4.13 (Request Entity Too Large) can be returned at any time by a server that does not currently have the resources to store blocks for a block-wise request payload transfer that it would intend to implement in an atomic fashion. (Note that a 4.13 response to a request that does not employ Block1 is a hint for the client to try sending Block1, and a 4.13 response with a smaller SZX in its Block1 option than requested is a hint to try a smaller SZX.)

The Block1 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations to the same resource. Starting a new block-wise sequence of requests to the same resource (before an old sequence from the same endpoint was finished) simply overwrites the context the server may still be keeping. (This is probably exactly what one wants in this case - the client may simply have restarted and lost its knowledge of the previous sequence.)

2.6. Combining Block-wise Transfers with the Observe Option

The Observe Option provides a way for a client to be notified about changes over time of a resource [I-D.ietf-core-observe]. Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. The following rules apply to the combination of block-wise transfers with notifications.

Observation relationships always apply to an entire resource; the Block2 option does not provide a way to observe a single block of a resource.

As with basic GET transfers, the client can indicate its desired block size in a Block2 Option in the GET request establishing or renewing the observation relationship. If the server supports block-wise transfers, it SHOULD take note of the block size and apply it as a maximum size to all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the entry in that list is updated by the server receiving a new GET request for the resource from the client).

When sending a 2.05 (Content) notification, the server only sends the first block of the representation. The client retrieves the rest of the representation as if it had caused this first response by a GET request, i.e., by using additional GET requests with Block2 options containing NUM values greater than zero. (This results in the transfer of the entire representation, even if only some of the blocks have changed with respect to a previous notification.)
As with other dynamically changing resources, to ensure that the blocks being reassembled are from the same version of the representation, the server SHOULD include an ETag option in each response, and the reassembling client MUST compare the ETag options (Section 2.4). Even more so than for the general case of Block2, clients that want to retrieve all blocks of a resource they have been notified about with a first block SHOULD strive to do so without undue delay.

See Section 3.4 for examples.

2.7. Combining Block1 and Block2

In PUT and particularly in POST exchanges, both the request body and the response body may be large enough to require the use of block-wise transfers. First, the Block1 transfer of the request body proceeds as usual. In the exchange of the last slice of this block-wise transfer, the response carries the first slice of the Block2 transfer (NUM is zero). To continue this Block2 transfer, the client continues to send requests similar to the requests in the Block1 phase, but leaves out the Block1 options and includes a Block2 request option with non-zero NUM.

Block2 transfers that retrieve the response body for a request that used Block1 MUST be performed in sequential order.

2.8. Combining Block2 with Multicast

A client can use the Block2 option in a multicast GET request with NUM = 0 to aid in limiting the size of the response.

Similarly, a response to a multicast GET request can use a Block2 option with NUM = 0 if the representation is large, or to further limit the size of the response.

In both cases, the client retrieves any further blocks using unicast exchanges; in the unicast requests, the client SHOULD heed any block size preferences indicated by the server in the response to the multicast request.

Other uses of the Block options in conjunction with multicast messages are for further study.

2.9. Response Codes

Two response codes are defined by this specification beyond those already defined in [RFC7252], and another response code is extended in its meaning.
2.9.1.  2.31 Continue

This new success status code indicates that the transfer of this block of the request body was successful and that the server encourages sending further blocks, but that a final outcome of the whole block-wise request cannot yet be determined. No payload is returned with this response code.

2.9.2.  4.08 Request Entity Incomplete

This new client error status code indicates that the server has not received the blocks of the request body that it needs to proceed. The client has not sent all blocks, not sent them in the order required by the server, or has sent them long enough ago that the server has already discarded them.

2.9.3.  4.13 Request Entity Too Large

In [RFC7252], section 5.9.2.9, the response code 4.13 (Request Entity Too Large) is defined to be like HTTP 413 "Request Entity Too Large". [RFC7252] also recommends that this response SHOULD include a Size1 Option (Section 4) to indicate the maximum size of request entity the server is able and willing to handle, unless the server is not in a position to make this information available.

The present specification allows the server to return this response code at any time during a Block1 transfer to indicate that it does not currently have the resources to store blocks for a transfer that it would intend to implement in an atomic fashion. It also allows the server to return a 4.13 response to a request that does not employ Block1 as a hint for the client to try sending Block1. Finally, a 4.13 response to a request with a Block1 option (control usage, see Section 2.3) where the response carries a smaller SZX in its Block1 option is a hint to try that smaller SZX.

2.10.  Caching Considerations

This specification attempts to leave a variety of implementation strategies open for caches, in particular those in caching proxies. E.g., a cache is free to cache blocks individually, but also could wait to obtain the complete representation before it serves parts of it. Partial caching may be more efficient in a cross-proxy (equivalent to a streaming HTTP proxy). A cached block (partial cached response) can be used in place of a complete response to satisfy a block-wise request that is presented to a cache. Note that different blocks can have different Max-Age values, as they are transferred at different times. A response with a block updates the freshness of the complete representation. Individual blocks can be
validated, and validating a single block validates the complete representation. A response with a Block1 Option in control usage with the M bit set invalidates cached responses for the target URI.

A cache or proxy that combines responses (e.g., to split blocks in a request or increase the block size in a response, or a cross-proxy) may need to combine 2.31 and 2.01/2.04 responses; a stateless server may be responding with 2.01 only on the first Block1 block transferred, which dominates any 2.04 responses for later blocks.

If-None-Match only works correctly on Block1 requests with (NUM=0) and MUST NOT be used on Block1 requests with NUM != 0.

3. Examples

This section gives a number of short examples with message flows for a block-wise GET, and for a PUT or POST. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and examples for the operation of the block size negotiation.

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

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 128 bytes of payload each, and third ACK contains between 1 and 128 bytes.
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.

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.

<table>
<thead>
<tr>
<th>CLIENT</th>
<th>SERVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON [MID=1234], GET, /status            -----&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;------- ACK [MID=1234], 2.05 Content, 2:0/1/128</td>
<td></td>
</tr>
<tr>
<td>CON [MID=1235], GET, /status, 2:2/0/64 -----&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;------- ACK [MID=1235], 2.05 Content, 2:2/1/64</td>
<td></td>
</tr>
<tr>
<td>CON [MID=1236], GET, /status, 2:3/0/64 -----&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;------- ACK [MID=1236], 2.05 Content, 2:3/1/64</td>
<td></td>
</tr>
<tr>
<td>CON [MID=1237], GET, /status, 2:4/0/64 -----&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;------- ACK [MID=1237], 2.05 Content, 2:4/1/64</td>
<td></td>
</tr>
<tr>
<td>CON [MID=1238], GET, /status, 2:5/0/64 -----&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;------- ACK [MID=1238], 2.05 Content, 2:5/0/64</td>
<td></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>CLIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON [MID=1234], GET, /status</td>
</tr>
<tr>
<td>&lt;------ ACK [MID=1234], 2.05 Content, 2:0/1/128</td>
</tr>
<tr>
<td>CON [MID=1235], GET, /status, 2:2/0/64</td>
</tr>
<tr>
<td>(timeout)</td>
</tr>
<tr>
<td>CON [MID=1235], GET, /status, 2:2/0/64</td>
</tr>
<tr>
<td>&lt;------ ACK [MID=1235], 2.05 Content, 2:2/1/64</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>CON [MID=1238], GET, /status, 2:5/0/64</td>
</tr>
<tr>
<td>&lt;------ ACK [MID=1238], 2.05 Content, 2:5/0/64</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>CLIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON [MID=1234], GET, /status</td>
</tr>
<tr>
<td>&lt;------ ACK [MID=1234], 2.05 Content, 2:0/1/128</td>
</tr>
<tr>
<td>CON [MID=1235], GET, /status, 2:2/0/64</td>
</tr>
<tr>
<td>///////////////////////////////////////////////////////////////////tent, 2:2/1/64</td>
</tr>
<tr>
<td>(timeout)</td>
</tr>
<tr>
<td>CON [MID=1235], GET, /status, 2:2/0/64</td>
</tr>
<tr>
<td>&lt;------ ACK [MID=1235], 2.05 Content, 2:2/1/64</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>CON [MID=1238], GET, /status, 2:5/0/64</td>
</tr>
<tr>
<td>&lt;------ ACK [MID=1238], 2.05 Content, 2:5/0/64</td>
</tr>
</tbody>
</table>
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.

CLIENT  

| CON [MID=1234], PUT, /options, 1:0/1/128  -----> |
| <------  ACK [MID=1234], 2.31 Continue, 1:0/1/128 |
| CON [MID=1235], PUT, /options, 1:1/1/128  -----> |
| <------  ACK [MID=1235], 2.31 Continue, 1:1/1/128 |
| CON [MID=1236], PUT, /options, 1:2/0/128  -----> |
| <------  ACK [MID=1236], 2.04 Changed, 1:2/0/128 |

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.
Finally, a server receiving a block-wise PUT or POST may want to indicate a smaller block size preference (Figure 9). In this case, the client SHOULD continue with a smaller block size; if it does, it MUST adjust the block number to properly count in that smaller size.

```
CLIENT                      SERVER

CON [MID=1234], PUT, /options, 1:0/1/128  ------>
|<------   ACK [MID=1234], 2.04 Changed, 1:0/0/128

CON [MID=1235], PUT, /options, 1:1/1/128  ------>
|<------   ACK [MID=1235], 2.04 Changed, 1:1/0/128

CON [MID=1236], PUT, /options, 1:2/0/128  ------>
|<------   ACK [MID=1236], 2.04 Changed, 1:2/0/128

Figure 8: Simple stateless block-wise PUT
```

3.3. Combining Block1 and Block2

Block options may be used in both directions of a single exchange. The following example demonstrates a block-wise POST request, resulting in a separate block-wise response.

```
CLIENT                      SERVER

CON [MID=1234], PUT, /options, 1:0/1/128  ------>
|<------   ACK [MID=1234], 2.31 Continue, 1:0/1/32

CON [MID=1235], PUT, /options, 1:4/1/32  ------>
|<------   ACK [MID=1235], 2.31 Continue, 1:4/1/32

CON [MID=1236], PUT, /options, 1:5/1/32  ------>
|<------   ACK [MID=1235], 2.31 Continue, 1:5/1/32

CON [MID=1237], PUT, /options, 1:6/0/32  ------>
|<------   ACK [MID=1236], 2.04 Changed, 1:6/0/32

Figure 9: Simple atomic block-wise PUT with negotiation
```
This model does provide for early negotiation input to the Block2 block-wise transfer, as shown below.

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

```
CLIENT    SERVER

CON [MID=1234], POST, /soap, 1:0/1/128 ------>
<------   ACK [MID=1234], 2.31 Continue, 1:0/1/128

CON [MID=1235], POST, /soap, 1:1/1/128 ------>
<------   ACK [MID=1235], 2.31 Continue, 1:1/1/128

CON [MID=1236], POST, /soap, 1:2/0/128, 2:0/0/64 ------>
<------   ACK [MID=1236], 2.04 Changed, 1:2/0/128, 2:0/1/64

CON [MID=1237], POST, /soap, 2:1/0/64 ------>
(no payload for requests with Block2 with NUM != 0)
<------   ACK [MID=1237], 2.04 Changed, 2:1/1/64

CON [MID=1238], POST, /soap, 2:2/0/64 ------>
<------   ACK [MID=1238], 2.04 Changed, 2:2/1/64

CON [MID=1239], POST, /soap, 2:3/0/64 ------>
<------   ACK [MID=1239], 2.04 Changed, 2:3/0/64
```

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

---
Figure 12: Observe sequence with block-wise response

(Note that the choice of token 0xfc in this examples is arbitrary; tokens are just shown in this example to illustrate that the requests...
for additional blocks cannot make use of the token of the Observation relationship. As a general comment on tokens, there is no other mention of tokens in this document, as block-wise transfers handle tokens like any other CoAP exchange. As usual the client is free to choose tokens for each exchange as it likes.)

In the following example, the client also uses early negotiation to limit the block size to 64 bytes.

CLIENT | SERVER

----->| GET
| HEADER: GET 0x41011636
| Token: 0xfb
| Uri-Path: status-icon
| Observe: (empty)
| Block2: 0/0/64

<------| 2.05
| HEADER: 2.05 0x61451636
| Token: 0xfb
| Block2: 0/1/64
| Observe: 62350
| ETag: 6f00f38e
| Max-Age: 60
| Payload: [64 bytes]

(Usual GET transfer left out)

...

(Notification of first block:)

<------| 2.05
| HEADER: 2.05 0x4145af9c
| Token: 0xfb
| Block2: 0/1/64
| Observe: 62354
| ETag: 6f00f392
| Payload: [64 bytes]

+- - ->| Header: 0x6000af9c

(Retrieval of remaining blocks)

----->| GET
| HEADER: GET 0x41011637
| Token: 0xfc
| Uri-Path: status-icon
| Block2: 1/0/64

<------| 2.05
| HEADER: 2.05 0x61451637
| Token: 0xfc
| Block2: 1/1/64
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 is already defined in [RFC7252] for 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 Size2 Option may be used for two purposes:

- o in a request, to ask the server to provide a size estimate along with the usual response ("size request"). For this usage, the value MUST be set to 0.

- o in a response carrying a Block2 Option, to indicate the current estimate the server has of the total size of the resource representation, measured in bytes ("size indication").

Similarly, the Size1 Option may be used for two purposes:

- o in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation, measured in bytes ("size indication").
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.

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>Size1</td>
<td>uint</td>
<td>0-4</td>
<td>(none)</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>Size2</td>
<td>uint</td>
<td>0-4</td>
<td>(none)</td>
</tr>
</tbody>
</table>

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]:

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Block2</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>27</td>
<td>Block1</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>28</td>
<td>Size2</td>
<td>[RFCXXXX]</td>
</tr>
</tbody>
</table>

Table 3: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [RFC7252]:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.31</td>
<td>Continue</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>4.08</td>
<td>Request Entity Incomplete</td>
<td>[RFCXXXX]</td>
</tr>
</tbody>
</table>

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.

A stateless server might be susceptible to an attack where the adversary sends a Block1 (e.g., PUT) block with a high block number: A naive implementation might exhaust its resources by creating a huge resource representation.

Misleading size indications may be used by an attacker to induce buffer overflows in poor implementations, for which the usual considerations apply.

7.1. Mitigating Resource Exhaustion Attacks

Certain block-wise requests may induce the server to create state, e.g. to create a snapshot for the block-wise GET of a fast-changing resource to enable consistent access to the same version of a resource for all blocks, or to create temporary resource representations that are collected until pressed into service by a final PUT or POST with the more bit unset. All mechanisms that induce a server to create state that cannot simply be cleaned up create opportunities for denial-of-service attacks. Servers SHOULD avoid being subject to resource exhaustion based on state created by untrusted sources. But even if this is done, the mitigation may cause a denial-of-service to a legitimate request when it is drowned out by other state-creating requests. Wherever possible, servers should therefore minimize the opportunities to create state for untrusted sources, e.g. by using stateless approaches.

Performing segmentation at the application layer is almost always better in this respect than at the transport layer or lower (IP fragmentation, adaptation layer fragmentation), for instance because there is application layer semantics that can be used for mitigation or because lower layers provide security associations that can prevent attacks. However, it is less common to apply timeouts and keepalive mechanisms at the application layer than at lower layers. Servers MAY want to clean up accumulated state by timing it out (cf. response code 4.08), and clients SHOULD be prepared to run block-wise transfers in an expedient way to minimize the likelihood of running into such a timeout.
7.2.  Mitigating Amplification Attacks

[RFC7252] discusses the susceptibility of CoAP end-points for use in amplification attacks.

A CoAP server can reduce the amount of amplification it provides to an attacker by offering large resource representations only in relatively small blocks. With this, e.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

8.  Acknowledgements

Much of the content of this draft is the result of discussions with the [RFC7252] authors, and via many CoRE WG discussions.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version. Esko Dijk reviewed a more recent version, leading to a number of further editorial improvements, a solution to the 4.13 ambiguity problem, and the section about combining Block and multicast. Markus Becker proposed getting rid of an ill-conceived default value for the Block2 and Block1 options. Peter Bigot insisted on a more systematic coverage of the options and response code.

Kepeng Li, Linyi Tian, and Barry Leiba wrote up an early version of the Size Option, which has informed this draft. Klaus Hartke wrote some of the text describing the interaction of Block2 with Observe. Matthias Kovatsch provided a number of significant simplifications of the protocol.

9.  References

9.1.  Normative References


9.2. Informative References


Authors’ Addresses

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

Phone: +49-421-218-63921
Email: cabo@tzi.org
Guidelines for HTTP-CoAP Mapping Implementations
draft-ietf-core-http-mapping-06

Abstract

This document provides reference information for implementing a proxy that performs translation between the HTTP protocol and the CoAP protocol, focusing on the reverse proxy case. It describes how a HTTP request is mapped to a CoAP request and how a CoAP response is mapped back to a HTTP response. Furthermore it defines a template for URI mapping and provides a set of guidelines for HTTP to CoAP protocol translation and related proxy implementations.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

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

1. Introduction ............................................. 3
2. Terminology .............................................. 4
3. HTTP-CoAP Reverse Proxy ................................. 5
4. Use Cases ................................................ 6
5. URI Mapping ............................................... 6
   5.1. URI Terminology ........................................ 7
   5.2. Default Mapping ........................................ 7
   5.2.1. Optional Scheme Omission ......................... 8
   5.2.2. Encoding Caveats .................................... 8
   5.3. URI Mapping Template ................................. 9
   5.3.1. Simple Form ....................................... 9
   5.3.2. Enhanced Form .................................... 11
   5.4. Discovery ............................................ 12
   5.4.1. Examples .......................................... 12
5. Media Type Mapping ....................................... 14
6. Media Type Mapping ....................................... 14
   6.1. Overview ............................................ 14
   6.2. Loose Media Type Mapping ............................ 15
   6.3. Media Type to Content Format Mapping Algorithm .... 16
   6.4. Content Transcoding .................................. 17
   6.4.1. General .......................................... 17
   6.4.2. CoRE Link Format .................................. 18
   6.4.3. Diagnostic Messages .............................. 18
7. Response Code Mapping ................................... 19
8. Additional Mapping Guidelines ......................... 21
   8.1. Caching and Congestion Control ..................... 21
   8.2. Cache Refresh via Observe ........................... 22
   8.3. Use of CoAP Blockwise Transfer ..................... 22
   8.4. Security Translation ................................ 23
   8.5. CoAP Multicast ...................................... 23
   8.6. Timeouts ............................................. 24
   8.7. Miscellaneous ...................................... 24
9. IANA Considerations ...................................... 25
10. Security Considerations ............................... 25
    10.1. Traffic Overflow ................................... 25
    10.2. Handling Secured Exchanges ....................... 26
    10.3. Proxy and CoAP Server Resource Exhaustion ....... 27
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;
o Section 7 describes how to map CoAP responses to HTTP responses;

o Section 8 describes additional mapping guidelines related to caching, congestion, timeouts and CoAP blockwise transfers;

o Section 10 discusses possible security impact of HTTP-CoAP protocol mapping.

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].

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/dereferencing 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](image-url)
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 in an unambiguous way 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:

\[
\begin{align*}
\text{cu} & = \text{coap-URI} ; \text{from RFC7252, Section 6.1} \\
\text{su} & = \text{coaps-URI} ; \text{from RFC7252, Section 6.2} \\
\text{tu} & = \text{cu} / \text{su}
\end{align*}
\]

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

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:
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:
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 an "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. Examples

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

On the HTTP side link information can be serialised 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:
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 HTTP/1.1
     Host: p.example.com

Res:  HTTP/1.1 200 OK
     Link: </hc>;rt="core.hc"
```

6.  Media Type Mapping

6.1.  Overview

An HC proxy needs to translate HTTP media types (Section 3.1.1.1 of [RFC7231]) and content encodings (Section 3.1.2.2 of [RFC7231]) into CoAP content formats (Section 12.3 of [RFC7252]) and vice versa.

Media type translation can happen in GET, PUT or POST requests going from HTTP to CoAP, and in 2.xx (i.e., successful) responses going from CoAP to HTTP. Specifically, PUT and POST need to map both the Content-Type and Content-Encoding HTTP headers into a single CoAP Content-Format option, whereas GET needs to map Accept and Accept-Encoding HTTP headers into a single CoAP Accept option. To generate the HTTP response, the CoAP Content-Format option is mapped back to a suitable HTTP Content-Type and Content-Encoding combination.

An HTTP request carrying a Content-Type and Content-Encoding combination which the HC proxy is unable to map to an equivalent CoAP Content-Format, SHALL elicit a 415 (Unsupported Media Type) response by the HC proxy.

If the HC proxy receives a CoAP response with a Content-Format that it does not recognise (for example because the value has been registered after the proxy has been implemented), then it is allowed to either return a HTTP entity without a Content-Type header, or examine the data to determine its type on the fly.
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, 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 cases, the "loose" media type mapping detailed in Section 6.2 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 unauthorised users to deplete or abuse systems and network resources.

6.2. 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 specialisations 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 generalised 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.

<table>
<thead>
<tr>
<th>Internet media type</th>
<th>Generalised media type</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/*+xml</td>
<td>application/xml</td>
</tr>
<tr>
<td>application/*+json</td>
<td>application/json</td>
</tr>
<tr>
<td>text/xml</td>
<td>application/xml</td>
</tr>
<tr>
<td>text/*</td>
<td>text/plain</td>
</tr>
<tr>
<td><em>/</em></td>
<td>application/octet-stream</td>
</tr>
</tbody>
</table>

Table 1: Media type generalisation 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 generalisations allowed.

6.3. 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 Table 9 defined in Section 12.3 of [RFC7252] plus, possibly, any locally defined extension of it. Optionally, the table and lookup mechanism described in Section 6.2 can be used if the implementation chooses so.

Note that the algorithm may have side effects on the associated representation (see also Section 6.4)
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 generalised 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**

### 6.4. Content Transcoding

#### 6.4.1. 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 6.3, 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 optimised
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. So whenever the HC proxy transcodes an application/XML to application/EXI in-band meta data could be lost. Therefore, the implementer should always carefully verify such lossy payload transformations before triggering the transcoding.

6.4.2. 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.

6.4.3. 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.

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).

<table>
<thead>
<tr>
<th>CoAP Response Code</th>
<th>HTTP Status Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.01 Created</td>
<td>201 Created</td>
<td>1</td>
</tr>
<tr>
<td>2.02 Deleted</td>
<td>200 OK</td>
<td>2</td>
</tr>
<tr>
<td>2.03 Valid</td>
<td>204 No Content</td>
<td>2</td>
</tr>
<tr>
<td>2.04 Changed</td>
<td>204 No Content</td>
<td>2</td>
</tr>
<tr>
<td>2.05 Content</td>
<td>200 OK</td>
<td></td>
</tr>
<tr>
<td>4.00 Bad Request</td>
<td>400 Bad Request</td>
<td></td>
</tr>
<tr>
<td>4.01 Unauthorized</td>
<td>401 Unauthorized</td>
<td>5</td>
</tr>
<tr>
<td>4.02 Bad Option</td>
<td>400 Bad Request</td>
<td>6</td>
</tr>
<tr>
<td>4.03 Forbidden</td>
<td>403 Forbidden</td>
<td></td>
</tr>
<tr>
<td>4.04 Not Found</td>
<td>404 Not Found</td>
<td></td>
</tr>
<tr>
<td>4.05 Method Not Allowed</td>
<td>405 Method Not Allowed</td>
<td>7</td>
</tr>
<tr>
<td>4.06 Not Acceptable</td>
<td>406 Not Acceptable</td>
<td></td>
</tr>
<tr>
<td>4.12 Precondition Failed</td>
<td>412 Precondition Failed</td>
<td></td>
</tr>
<tr>
<td>4.13 Request Ent. Too Large</td>
<td>413 Request Repr. Too Large</td>
<td></td>
</tr>
<tr>
<td>4.15 Unsupported Media Type</td>
<td>415 Unsupported Media Type</td>
<td></td>
</tr>
<tr>
<td>5.00 Internal Server Error</td>
<td>500 Internal Server Error</td>
<td></td>
</tr>
<tr>
<td>5.01 Not Implemented</td>
<td>501 Not Implemented</td>
<td></td>
</tr>
<tr>
<td>5.02 Bad Gateway</td>
<td>502 Bad Gateway</td>
<td></td>
</tr>
<tr>
<td>5.03 Service Unavailable</td>
<td>503 Service Unavailable</td>
<td>8</td>
</tr>
<tr>
<td>5.04 Gateway Timeout</td>
<td>504 Gateway Timeout</td>
<td></td>
</tr>
<tr>
<td>5.05 Proxying Not Supported</td>
<td>502 Bad Gateway</td>
<td>9</td>
</tr>
</tbody>
</table>

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 recognised 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. HTTP code 405 (Method Not Allowed) MUST include an "Allow" response-header field (Section 7.4.1 of [RFC7231]). However, a CoAP response does not include information about which methods are allowed on the resource. Therefore, if the proxy does not have further information about which methods are allowed on the resource it SHOULD include an empty field value in the Allow header field. The intended interpretation of an empty Allow in
this case is "resource temporarily allows no methods" which complies fully to [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. Further 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 the refresh of 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 $T_R < 2T_C$ 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
o preset to a known IP MTU value, or
o set to a known Path MTU value.

The value BLOCKWISE_THRESHOLD, or the parameters from which it is calculated, should be configurable in a proxy implementation. The maximum block size the proxy will attempt to use in CoAP requests should also be configurable.

The HC proxy SHOULD detect CoAP 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.

However, details of 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 supporting 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

$T = \text{MAX\_RTT} + \text{MAX\_SERVER\_RESPONSE\_DELAY}$

where MAX\_RTT is defined in [RFC7252] and MAX\_SERVER\_RESPONSE\_DELAY is defined in [RFC7390]. An exception to this rule occurs when the 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

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.

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 accessed exclusively in a secure manner.

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 whitelist multicast resources/authorities that are allowed to be dereferenced. 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 whitelist 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 [RFC7252]. 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]

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

[RFC2616]  Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext

[RFC3040]  Cooper, I., Melve, I., and G. Tomlinson, "Internet Web

[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-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;
o Addressed Ticket #376 (CoAP 4.05 response can’t be translated to HTTP 405 by HC proxy) in Section 7 by use of empty ‘Allow’ header;

o Removed details on the pros and cons of HC proxy placement options;

o Addressed review comments of Carsten Bormann;

o Clarified failure in mapping of HTTP Accept headers (Section 6.3);

o Clarified detection of CoAP servers not supporting blockwise (Section 8.3);

o Changed CoAP request timeout min value to MAX_RTT + MAX_SERVER_RESPONSE_DELAY (Section 8.6);

o Added security section item (Section 10.3) related to use of CoAP blockwise transfers;

o Many editorial improvements.

Changes from ietf-04 to ietf-05:

o Addressed Ticket #366 (Mapping of CoRE Link Format payloads to be valid in HTTP Domain?) in Section 6.3.3.2 (Content Transcoding - CORE Link Format);

o Addressed Ticket #375 (Add requirement on mapping of CoAP diagnostic payload) in Section 6.3.3.3 (Content Transcoding - Diagnostic Messages);

o Addressed comment from Yusuke (http://www.ietf.org/mail-archive/web/core/current/msg05491.html) in Section 6.3.3.1 (Content Transcoding - General);

o Various editorial improvements.

Changes from ietf-03 to ietf-04:

o Expanded use case descriptions in Section 4;

o Fixed/enhanced discovery examples in Section 5.4.1;

o Addressed Ticket #365 (Add text on media type conversion by HTTP-CoAP proxy) in new Section 6.3.1 (Generalized media type mapping) and new Section 6.3.2 (Content translation);
o Updated HTTPBis WG draft references to recently published RFC numbers.

o Various editorial improvements.

Changes from ietf-02 to ietf-03:

o Closed Ticket #351 "Add security implications of proposed default HTTP-CoAP URI mapping";

o Closed Ticket #363 "Remove CoAP scheme in default HTTP-CoAP URI mapping";

o Closed Ticket #364 "Add discovery of HTTP-CoAP mapping resource(s)".

Changes from ietf-01 to ietf-02:

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

Changes from ietf-00 to ietf-01:

o Added URI mapping proposals to Section 4 as per the Email proposals to WG mailing list from Esko.

Authors’ Addresses

Angelo P. Castellani
University of Padova
Via Gradenigo 6/B
Padova  35131
Italy

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas  02420
Finland

Email: salvatore.loreto@ericsson.com
Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal H3A 3G4
Canada

Phone: +1 514 585 0761
Email: Akbar.Rahman@InterDigital.com

Thomas Fossati
Alcatel-Lucent
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: thomas.fossati@alcatel-lucent.com

Esko Dijk
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
The Netherlands

Email: esko.dijk@philips.com
Abstract

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 format (RFC7159).

This specification defines a common format for representing Web links in JSON format.
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].

Outside of constrained environments, it may also be useful to represent the same collections of Web links in the widely used JSON format [RFC7159]. When converting between these two formats, as usual, there are many little decisions that have to be made. If left without guidance, it is likely that a number of slightly incompatible dialects will emerge.

This specification defines a common format for representing CoRE Web Linking in JSON format.

Note that there is a separate question on how to represent Web links 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]
  - 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", "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.

2. Web Links in JSON

The objective of the JSON mapping 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.

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.

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 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 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 the most recent version of link-format 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]).

(TBD: Should we do something special with the "hosts" relation? Should we include an anchor where the link-format does not explicitly set one?)

2.1. 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 1: Example from page 15 of [RFC6690]

becomes

```
```

Bormann Expires January 5, 2015 [Page 4]
3. 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 Section 6 (RFC7159).

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
4. Security Considerations

The security considerations of [RFC6690] apply.

(TBD.)

5. Acknowledgements

(TBD.)

6. References

6.1. Normative References


6.2. Informative References

[I-D.pbryan-zyp-json-ref]


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.)

Author’s Address

Carsten Bormann
Universitaet Bremen T2I
Postfach 330440
Bremen  D-28359
Germany

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

The Constrained Application Protocol (CoAP) is a RESTful application protocol for constrained nodes and networks. The state of a resource on a CoAP server can change over time. This document specifies a simple protocol extension for CoAP that enables CoAP clients to "observe" resources, i.e., to retrieve a representation of a resource and keep this representation updated by the server over a period of time. The protocol follows a best-effort approach for sending new representations to clients and provides eventual consistency between the state observed by each client and the actual resource state at the server.

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 30, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ........................................... 3
   1.1. Background ..................................... 3
   1.2. Protocol Overview ............................... 3
   1.3. Consistency Model ............................... 5
   1.4. Observable Resources ............................ 6
   1.5. Requirements Notation ............................ 7
2. The Observe Option ..................................... 7
3. Client-side Requirements .............................. 8
   3.1. Request ......................................... 8
   3.2. Notifications ................................... 8
   3.3. Caching ........................................ 9
   3.4. Reordering ..................................... 10
   3.5. Transmission ................................... 11
   3.6. Cancellation ................................... 11
4. Server-side Requirements .............................. 12
   4.1. Request ......................................... 12
   4.2. Notifications ................................... 13
   4.3. Caching ........................................ 13
   4.4. Reordering ..................................... 14
   4.5. Transmission ................................... 15
5. Intermediaries ........................................ 18
6. Web Linking ........................................... 19
7. Security Considerations ............................... 19
8. IANA Considerations .................................. 20
9. Acknowledgements ..................................... 20
10. References ........................................... 21
    10.1. Normative References ........................... 21
    10.2. Informative References ........................ 21
Appendix A. Examples ...................................... 22
    A.1. Client/Server Examples .......................... 22
    A.2. Proxy Examples ................................ 26
Appendix B. Changelog .................................... 28
1. Introduction

1.1. Background

The Constrained Application Protocol (CoAP) [RFC7252] is intended to provide RESTful services [REST] 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 [RFC7228].

The model of REST is that of a client exchanging representations of resources with a server, where a representation captures the current or intended state of a resource and the server is the authority for representations of the resources in its namespace. A client interested in the state of a resource initiates a request to the server; the server then returns a response with a representation of the resource that is current at the time of the request.

This model does not work well when a client is interested in having a current representation of a resource over a period of time. Existing approaches from HTTP, such as repeated polling or HTTP long polling [RFC6202], generate significant complexity and/or overhead and thus are less applicable in a constrained environment.

The protocol specified in this document extends the CoAP core protocol with a mechanism for a CoAP client to "observe" a resource on a CoAP server: the client retrieves a representation of the resource and requests this representation be updated by the server as long as the client is interested in the resource.

The protocol keeps the architectural properties of REST. It enables high scalability and efficiency through the support of caches and proxies. There is no intention, though, to solve the full set of problems that the existing HTTP solutions solve, or to replace publish/subscribe networks that solve a much more general problem [RFC5989].

1.2. Protocol Overview

The protocol is based on the well-known observer design pattern [GOF]. In this design pattern, components called "observers" register at a specific, known provider called the "subject" that they are interested in being notified whenever the subject undergoes a change in state. The subject is responsible for administering its list of registered observers. If multiple subjects are of interest to an observer, the observer must register separately for all of them.
The observer design pattern is realized in CoAP as follows:

Subject: In the context of CoAP, the subject is a resource in the namespace of a CoAP server. The state of the resource can change over time, ranging from infrequent updates to continuous state transformations.

Observer: An observer is a CoAP client that is interested in having a current representation of the resource at any given time.

Registration: A client registers its interest in a resource by initiating an extended GET request to the server. In addition to returning a representation of the target resource, this request causes the server to add the client to the list of observers of the resource.

Notification: Whenever the state of a resource changes, the server notifies each client in the list of observers of the resource. Each notification is an additional CoAP response sent by the server in reply to the single extended GET request, and includes a complete, updated representation of the new resource state.

Figure 2 below shows an example of a CoAP client registering its interest in a resource and receiving three notifications: the first with the current state upon registration, and then two upon changes to the resource state. Both the registration request and the notifications are identified as such by the presence of the Observe Option defined in this document. In notifications, the Observe Option additionally provides a sequence number for reordering detection. All notifications carry the token specified by the client, so the client can easily correlate them to the request.
A client remains on the list of observers as long as the server can determine the client’s continued interest in the resource. The server may send a notification in a confirmable CoAP message to request an acknowledgement by the client. When the client deregisters, rejects a notification, or the transmission of a notification times out after several transmission attempts, the client is considered no longer interested and is removed by the server from the list of observers.

1.3. Consistency Model

While a client is in the list of observers of a resource, the goal of the protocol is to keep the resource state observed by the client as closely in sync with the actual state at the server as possible.

It cannot be avoided that the client and the server become out of sync at times: First, there is always some latency between the change of the resource state and the receipt of the notification. Second, CoAP messages with notifications can get lost, which will cause the client to assume an old state until it receives a new notification.
And third, the server may erroneously come to the conclusion that the client is no longer interested in the resource, which will cause the server to stop sending notifications and the client to assume an old state until it eventually registers its interest again.

The protocol addresses this issue as follows:

- It follows a best-effort approach for sending the current representation to the client after a state change: Clients should see the new state after a state change as soon as possible, and they should see as many states as possible. This is limited by congestion control, however, so a client cannot rely on observing every single state that a resource might go through.

- It labels notifications with a maximum duration up to which it is acceptable for the observed state and the actual state to be out of sync. When the age of the notification received reaches this limit, the client cannot use the enclosed representation until it receives a new notification.

- It is designed on the principle of eventual consistency: The protocol guarantees that, if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state.

1.4. Observable Resources

A CoAP server is the authority for determining under what conditions resources change their state and thus when observers are notified of new resource states. The protocol does not offer explicit means for setting up triggers or thresholds; it is up to the server to expose observable resources that change their state in a way that is useful in the application context.

For example, a CoAP server with an attached temperature sensor could expose one or more of the following resources:

- `<coap://server/temperature>`, which changes its state every few seconds to a current reading of the temperature sensor;

- `<coap://server/temperature/felt>`, which changes its state to "COLD" whenever the temperature reading drops below a certain pre-configured threshold, and to "WARM" whenever the reading exceeds a second, slightly higher threshold;

- `<coap://server/temperature/critical?above=42>`, which changes its state based on the client-specified parameter value: every few seconds to the current temperature reading if the temperature...
exceeds the threshold, or to "OK" when the reading drops below;

- \(<\text{coap://server/?query=select+avg(temperature)+from+Sensor.window:time(30sec)}>\), which accepts expressions of arbitrary complexity and changes its state accordingly.

Thus, by designing CoAP resources that change their state on certain conditions, it is possible to update the client only when these conditions occur instead of supplying it continuously with raw sensor data. By parameterizing resources, this is not limited to conditions defined by the server, but can be extended to arbitrarily complex queries specified by the client. The application designer therefore can choose exactly the right level of complexity for the application envisioned and devices involved, and is not constrained to a "one size fits all" mechanism built into the protocol.

### 1.5. Requirements 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 RFC 2119 [RFC2119].

### 2. The Observe Option

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td></td>
<td>Observe</td>
<td>uint</td>
<td>0-3 B</td>
<td>(none)</td>
</tr>
</tbody>
</table>

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Table 1: The Observe Option

The Observe Option, when present in a request, extends the GET method so it does not only retrieve a current representation of the target resource, but also requests the server to add or remove an entry in the list of observers of the resource, where the entry consists of the client endpoint and the token specified in the request.

'register' (0) adds the entry to the list, if not present;

deregister' (1) removes the entry from the list, if present.

The Observe Option is not critical for processing the request. If the server is unwilling or unable to add a new entry to the list of observers, then the request falls back to a normal GET request, and the response does not include the Observe Option.
In a response, the Observe Option identifies the message as a notification. This implies that the server has added an entry with the client endpoint and request token to the list of observers and that it will notify the client of changes to the resource state. The option value is a 24-bit sequence number for reordering detection (see Section 3.4 and Section 4.4).

The value of the Observe Option is encoded as an unsigned integer in network byte order using a variable number of bytes (‘uint’ option format); see Section 3.2 of RFC 7252 [RFC7252].

The Observe Option is not part of the cache-key: a cacheable response obtained with an Observe Option in the request can be used to satisfy a request without an Observe Option, and vice versa. When a stored response with an Observe Option is used to satisfy a normal GET request, the option MUST be removed before the response is returned.

3. Client-side Requirements

3.1. Request

A client registers its interest in a resource by issuing a GET request with an Observe Option set to ‘register’ (0). If the server returns a 2.xx response that includes an Observe Option as well, the server has successfully added an entry with the client endpoint and request token to the list of observers of the target resource and the client will be notified of changes to the resource state.

Like a fresh response can be used to satisfy a request without contacting the server, the stream of updates resulting from one observation request can be used to satisfy another (observation or normal GET) request if the target resource is the same. A client MUST aggregate such requests and MUST NOT register more than once for the same target resource. The target resource is identified by all options in the request that are part of the cache-key. This includes, for example, the full request URI and the Accept Option.

3.2. Notifications

Notifications are additional responses sent by the server in reply to the single extended GET request that created the registration. Each notification includes the token specified by the client in the request. The only difference between a notification and a normal response is the presence of the Observe Option.

Notifications typically have a 2.05 (Content) response code. They include an Observe Option with a sequence number for reordering detection (see Section 3.4), and a payload in the same Content-Format
as the initial response. If the client included one or more ETag Options in the GET request (see Section 3.3), notifications can have a 2.03 (Valid) response code rather than a 2.05 (Content) response code. Such notifications include an Observe Option with a sequence number but no payload.

In the event that the resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server sends a notification with an appropriate response code (such as 4.04 Not Found) and removes the client’s entry from the list of observers of the resource. Non-2.xx responses do not include an Observe Option.

3.3. Caching

As notifications are just additional responses to a GET request, notifications partake in caching as defined in Section 5.6 of RFC 7252 [RFC7252]. Both the freshness model and the validation model are supported.

3.3.1. Freshness

A client MAY store a notification like a response in its cache and use a stored notification that is fresh without contacting the server. Like a response, a notification is considered fresh while its age is not greater than the value indicated by the Max-Age Option (and no newer notification/response has been received).

The server will do its best to keep the resource state observed by the client as closely in sync with the actual state as possible. However, a client cannot rely on observing every single state that a resource might go through. For example, if the network is congested or the state changes more frequently than the network can handle, the server can skip notifications for any number of intermediate states.

The server uses the Max-Age Option to indicate an age up to which it is acceptable that the observed state and the actual state are inconsistent. If the age of the latest notification becomes greater than its indicated Max-Age, then the client MUST NOT assume that the enclosed representation reflects the actual resource state.

To make sure it has a current representation and/or to re-register its interest in a resource, a client MAY issue a new GET request with the same token as the original at any time. All options MUST be identical to those in the original request, except for the set of ETag Options. It is RECOMMENDED that the client does not issue the request while it still has a fresh notification/response for the resource in its cache. Additionally, the client SHOULD at least wait
for a random amount of time between 5 and 15 seconds after Max-Age expired to reduce collisions with other clients.

3.3.2. Validation

When a client has one or more notifications stored in its cache for a resource, it can use the ETag Option in the GET request to give the server an opportunity to select a stored notification to be used.

The client MAY include an ETag Option for each stored response that is applicable in the GET request. Whenever the observed resource changes to a representation identified by one of the ETag Options, the server can select a stored response by sending a 2.03 (Valid) notification with an appropriate ETag Option instead of a 2.05 (Content) notification.

A client implementation needs to keep all candidate responses in its cache until it is no longer interested in the target resource or it re-registers with a new set of entity-tags.

3.4. Reordering

Messages with notifications can arrive in a different order than they were sent. Since the goal is to keep the observed state as closely in sync with the actual state as possible, a client MUST consider the notification that was sent most recently as the freshest, regardless of the order of arrival.

To provide an order among notifications for the client, the server sets the value of the Observe Option in each notification to the 24 least-significant bits of a strictly increasing sequence number. An incoming notification was sent more recently than the freshest notification so far when one of the following conditions is met:

\[
(V1 < V2 \text{ and } V2 - V1 < 2^{23}) \text{ or } \\
(V1 > V2 \text{ and } V1 - V2 > 2^{23}) \text{ or } \\
(T2 > T1 + 128 \text{ seconds})
\]

where \(V1\) is the value of the Observe Option in the freshest notification so far, \(V2\) the value of the Observe Option in the incoming notification, \(T1\) a client-local timestamp for the freshest notification so far, and \(T2\) a client-local timestamp for the incoming notification.
Design Note: The first two conditions verify that V1 is less than V2 in 24-bit serial number arithmetic [RFC1982]. The third condition ensures that the time elapsed between the two incoming messages is not so large that the difference between V1 and V2 has become larger than the largest integer that it is meaningful to add to a 24-bit serial number; in other words, after 128 seconds have elapsed without any notification, a client does not need to check the sequence numbers to assume that an incoming notification was sent more recently than the freshest notification it has received so far.

The duration of 128 seconds was chosen as a nice round number greater than MAX_LATENCY (Section 4.8.2 of RFC 7252 [RFC7252]).

3.5. Transmission

A notification can be confirmable or non-confirmable, i.e., it can be sent in a confirmable or a non-confirmable message. The message type used for a notification is independent of the type used for the request and of any previous notification.

If a client does not recognize the token in a confirmable notification, it MUST NOT acknowledge the message and SHOULD reject it with a Reset message; otherwise, the client MUST acknowledge the message as usual. In the case of a non-confirmable notification, rejecting the message with a Reset message is OPTIONAL.

An acknowledgement message signals to the server that the client is alive and interested in receiving further notifications; if the server does not receive an acknowledgement in reply to a confirmable notification, it will assume that the client is no longer interested and will eventually remove the associated entry from the list of observers.

3.6. Cancellation

A client that is no longer interested in receiving notifications for a resource can simply "forget" the observation. When the server then sends the next notification, the client will not recognize the token in the message and thus will return a Reset message. This causes the server to remove the associated entry from the list of observers. The entries in lists of observers are effectively "garbage collected" by the server.
Implementation Note: Due to potential message loss, the Reset message may not reach the server. The client may therefore have to reject multiple notifications, each with one Reset message, until the server finally removes the associated entry from the list of observers and stops sending notifications.

In some circumstances, it may be desirable to cancel an observation and release the resources allocated by the server to it more eagerly. In this case, a client MAY explicitly deregister by issuing a GET request which has the Token field set to the token of the observation to be cancelled and includes an Observe Option with the value set to ‘deregister’ (1). All other options MUST be identical to those in the registration request, except for the set of ETag Options. When the server receives such a request, it will remove any matching entry from the list of observers and process the GET request as usual.

4. Server-side Requirements

4.1. Request

A GET request with an Observe Option set to ‘register’ (0) requests the server not only to return a current representation of the target resource, but also to add the client to the list of observers of that resource. Upon success, the server returns a current representation of the resource and MUST keep this representation updated (as described in Section 1.3) as long as the client is on the list of observers.

The entry in the list of observers is keyed by the client endpoint and the token specified by the client in the request. If an entry with a matching endpoint/token pair is already present in the list (which, for example, happens when the client wishes to reinforce its interest in a resource), the server MUST NOT add a new entry but MUST replace or update the existing one.

A server that is unable or unwilling to add a new entry to the list of observers of a resource MAY silently ignore the registration request and process the GET request as usual. The resulting response MUST NOT include an Observe Option, the absence of which signals to the client that it will not be notified of changes to the resource and, e.g., needs to poll the resource for its state instead.

If the Observe Option in a request is set to any other value than ‘register’ (0), then the server MUST remove any entry with a matching endpoint/token pair from the list of observers and process the GET request as usual. The resulting response MUST NOT include an Observe Option.
4.2. Notifications

A client is notified of changes to the resource state by additional responses sent by the server in reply to the GET request. Each such notification response (including the initial response) MUST echo the token specified by the client in the GET request. If there are multiple entries in the list of observers, the order in which the clients are notified is not defined; the server is free to use any method to determine the order.

A notification SHOULD have a 2.05 (Content) or 2.03 (Valid) response code. However, in the event that the state of a resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server SHOULD notify the client by sending a notification with an appropriate response code (such as 4.04 Not Found) and subsequently MUST remove the associated entry from the list of observers of the resource.

The Content-Format specified in a 2.xx notification MUST be the same as the one used in the initial response to the GET request. If the server is unable to continue sending notifications in this format, it SHOULD send a notification with a 4.06 (Not Acceptable) response code and subsequently MUST remove the associated entry from the list of observers of the resource.

A 2.xx notification MUST include an Observe Option with a sequence number as specified in Section 4.4 below; a non-2.xx notification MUST NOT include an Observe Option.

4.3. Caching

As notifications are just additional responses sent by the server in reply to a GET request, they are subject to caching as defined in Section 5.6 of RFC 7252 [RFC7252].

4.3.1. Freshness

After returning the initial response, the server MUST keep the resource state that is observed by the client as closely in sync with the actual resource state as possible.

Since becoming out of sync at times cannot be avoided, the server MUST indicate for each representation an age up to which it is acceptable that the observed state and the actual state are inconsistent. This age is application-dependent and MUST be specified in notifications using the Max-Age Option.
When the resource does not change and the client has a current representation, the server does not need to send a notification. However, if the client does not receive a notification, the client cannot tell if the observed state and the actual state are still in sync. Thus, when the age of the latest notification becomes greater than its indicated Max-Age, the client no longer has a usable representation of the resource state. The server MAY wish to prevent that by sending a new notification with the unchanged representation and a new Max-Age just before the Max-Age indicated earlier expires.

4.3.2. Validation

A client can include a set of entity-tags in its request using the ETag Option. When a resource changes its state and the origin server is about to send a 2.05 (Content) notification, then, whenever that notification has an entity-tag in the set of entity-tags specified by the client, the server MAY send a 2.03 (Valid) response with an appropriate ETag Option instead.

4.4. Reordering

Because messages can get reordered, the client needs a way to determine if a notification arrived later than a newer notification. For this purpose, the server MUST set the value of the Observe Option of each notification it sends to the 24 least-significant bits of a strictly increasing sequence number. The sequence number MAY start at any value and MUST NOT increase so fast that it increases by more than $2^{23}$ within less than 256 seconds.

The sequence number selected for a notification MUST be greater than that of any preceding notification sent to the same client with the same token for the same resource. The value of the Observe Option MUST be current at the time of transmission; if a notification is retransmitted, the server MUST update the value of the option to the sequence number that is current at that time before retransmission.

Implementation Note: A simple implementation that satisfies the requirements is to obtain a timestamp from a local clock. The sequence number then is the timestamp in ticks, where 1 tick = $(256 \text{ seconds})/(2^{23}) = 30.52 \text{ microseconds}$. It is not necessary that the clock reflects the current time/date.

Another valid implementation is to store a 24-bit unsigned integer variable per resource and increment this variable each time the resource undergoes a change of state (provided that the resource changes its state less than $2^{23}$ times in the first 256 seconds after every state change). This removes the need to update the value of the Observe Option on retransmission when the resource
Design Note: The choice of a 24-bit option value and a time span of 256 seconds theoretically allows for a notification rate of up to 65536 notifications per second. Constrained nodes often have rather imprecise clocks, though, and inaccuracies of the client and server side may cancel out or add in effect. Therefore, the maximum notification rate is reduced to 32768 notifications per second. This is still well beyond the highest known design objective of around 1 kHz (most CoAP applications will be several orders of magnitude below that), but allows total clock inaccuracies of up to -50/+100 %.

4.5. Transmission

A notification can be sent in a confirmable or a non-confirmable message. The message type used is typically application-dependent and may be determined by the server for each notification individually.

For example, for resources that change in a somewhat predictable or regular fashion, notifications can be sent in non-confirmable messages; for resources that change infrequently, notifications can be sent in confirmable messages. The server can combine these two approaches depending on the frequency of state changes and the importance of individual notifications.

A server MAY choose to skip sending a notification if it knows that it will send another notification soon, for example, when the state of a resource is changing frequently. It also MAY choose to send more than one notification for the same resource state. However, above all, the server MUST ensure that a client in the list of observers of a resource eventually observes the latest state if the resource does not undergo a new change in state.

For example, when state changes occur in bursts, the server can skip some notifications, send the notifications in non-confirmable messages, and make sure that the client observes the latest state change by repeating the last notification in a confirmable message when the burst is over.

The client’s acknowledgement of a confirmable notification signals that the client is interested in receiving further notifications. If a client rejects a confirmable or non-confirmable notification with a Reset message, or if the last attempt to retransmit a confirmable notification times out, then the client is considered no longer interested and the server MUST remove the associated entry from the list of observers.
Implementation Note: To properly process a Reset message that rejects a non-confirmable notification, a server needs to remember the message IDs of the non-confirmable notifications it sends. This may be challenging for a server with constrained resources. However, since Reset messages are transmitted unreliably, the client must be prepared that its Reset messages aren’t received by the server. A server thus can always pretend that a Reset message rejecting a non-confirmable notification was lost. If a server does this, it could accelerate cancellation by sending the following notifications to that client in confirmable messages.

A server that transmits notifications mostly in non-confirmable messages MUST send a notification in a confirmable message instead of a non-confirmable message at least every 24 hours. This prevents a client that went away or is no longer interested from remaining in the list of observers indefinitely.

4.5.1. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2 of RFC 7252 [RFC7252] and the limitations in Section 4.7 of RFC 7252 [RFC7252]. However, CoAP places the responsibility of congestion control for simple request/response interactions only on the clients: rate limiting request transmission implicitly controls the transmission of the responses. When a single request yields a potentially infinite number of notifications, additional responsibility needs to be placed on the server.

In order not to cause congestion, servers MUST strictly limit the number of simultaneous outstanding notifications/responses that they transmit to a given client to NSTART (1 by default; see Section 4.7 of RFC 7252 [RFC7252]). An outstanding notification/response is either a confirmable message for which an acknowledgement has not yet been received and whose last retransmission attempt has not yet timed out, or a non-confirmable message for which the waiting time that results from the following rate limiting rules has not yet elapsed.

The server SHOULD NOT send more than one non-confirmable notification per round-trip time (RTT) to a client on average. If the server cannot maintain an RTT estimate for a client, it SHOULD NOT send more than one non-confirmable notification every 3 seconds, and SHOULD use an even less aggressive rate when possible (see also Section 3.1.2 of RFC 5405 [RFC5405]).

Further congestion control optimizations and considerations are expected in the future with advanced CoAP congestion control mechanisms.
4.5.2. Advanced Transmission

The state of an observed resource may change while the number of the number of simultaneous outstanding notifications/responses to a client on the list of observers is greater than or equal to NSTART. In this case, the server cannot notify the client of the new resource state immediately but has to wait for an outstanding notification/response to complete first.

If there exists an outstanding notification/response that the server transmits to the client and that pertains to the changed resource, then it is desirable for the server to stop working towards getting the representation of the old resource state to the client, and to start transmitting the current representation to the client instead, so the resource state observed by the client stays closer in sync with the actual state at the server.

For this purpose, the server MAY optimize the transmission process by aborting the transmission of the old notification (but not before the current transmission attempt completed) and starting a new transmission for the new notification (but with the retransmission timer and counter of the aborted transmission retained).

In more detail, a server MAY supersede an outstanding transmission that pertains to an observation as follows:

1. Wait for the current (re-)transmission attempt to be acknowledged, rejected or to time out (confirmable transmission); or wait for the waiting time to elapse or the transmission to be rejected (non-confirmable transmission).

2. If the transmission is rejected or it was the last attempt to retransmit a notification, remove the associated entry from the list of observers of the observed resource.

3. If the entry is still in the list of observers, start to transmit a new notification with a representation of the current resource state. Should the resource have changed its state more than once in the meantime, the notifications for the intermediate states are silently skipped.

4. The new notification is transmitted with a new Message ID and the following transmission parameters: If the previous (re-)transmission attempt timed out, retain its transmission parameters, increment the retransmission counter and double the timeout; otherwise, initialize the transmission parameters as usual (see Section 4.2 of RFC 7252 [RFC7252]).
It is possible that the server later receives an acknowledgement for a confirmable notification that it superseded this way. Even though this does not signal consistency, it is valuable in that it signals the client’s further interest in the resource. The server therefore should avoid inadvertently removing the associated entry from the list of observers.

5. Intermediaries

A client may be interested in a resource in the namespace of a server that is reached through a chain of one or more CoAP intermediaries. In this case, the client registers its interest with the first intermediary towards the server, acting as if it was communicating with the server itself, as specified in Section 3. It is the task of this intermediary to provide the client with a current representation of the target resource and to keep the representation updated upon changes to the resource state, as specified in Section 4.

To perform this task, the intermediary SHOULD make use of the protocol specified in this document, taking the role of the client and registering its own interest in the target resource with the next hop towards the server. If the response returned by the next hop doesn’t include an Observe Option, the intermediary MAY resort to polling the next hop or MAY itself return a response without an Observe Option.

The communication between each pair of hops is independent; each hop in the server role MUST determine individually how many notifications to send, of which message type, and so on. Each hop MUST generate its own values for the Observe Option in notifications, and MUST set the value of the Max-Age Option according to the age of the local current representation.

If two or more clients have registered their interest in a resource with an intermediary, the intermediary MUST register itself only once with the next hop and fan out the notifications it receives to all registered clients. This relieves the next hop from sending the same notifications multiple times and thus enables scalability.

An intermediary is not required to act on behalf of a client to observe a resource; an intermediary MAY observe a resource, for example, just to keep its own cache up to date.

See Appendix A.2 for examples.
6. Web Linking

A web link [RFC5988] to a resource accessible over CoAP (for example, in a link-format document [RFC6690]) MAY include the target attribute "obs".

The "obs" attribute, when present, is a hint indicating that the destination of a link is useful for observation and thus, for example, should have a suitable graphical representation in a user interface. Note that this is only a hint; it is not a promise that the Observe Option can actually be used to perform the observation. A client may need to resort to polling the resource if the Observe Option is not returned in the response to the GET request.

A value MUST NOT be given for the "obs" attribute; any present value MUST be ignored by parsers. The "obs" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by parsers.

7. Security Considerations

The security considerations in Section 11 of the CoAP specification [RFC7252] apply.

Observing resources can dramatically increase the negative effects of amplification attacks. That is, not only can notifications messages be much larger than the request message, but the nature of the protocol can cause a significant number of notifications to be generated. Without client authentication, a server therefore MUST strictly limit the number of notifications that it sends between receiving acknowledgements that confirm the actual interest of the client in the data; i.e., any notifications sent in non-confirmable messages MUST be interspersed with confirmable messages. (An attacker may still spoof the acknowledgements if the confirmable messages are sufficiently predictable.)

The protocol follows a best-effort approach for keeping the state observed by a client and the actual resource state at a server in sync. This may have the client and the server become out of sync at times. Depending on the sensitivity of the observed resource, operating on an old state might be a security threat. The client therefore must be careful not to use a representation after its Max-Age expires, and the server must set the Max-Age Option to a sensible value.

As with any protocol that creates state, attackers may attempt to exhaust the resources that the server has available for maintaining the list of observers for each resource. Servers may want to apply
access controls to this creation of state. As degraded behavior, the server can always fall back to processing the request as a normal GET request (without an Observe Option) if it is unwilling or unable to add a client to the list of observers of a resource, including if system resources are exhausted or nearing exhaustion.

Intermediaries must be careful to ensure that notifications cannot be employed to create a loop. A simple way to break any loops is to employ caches for forwarding notifications in intermediaries.

Resources can be observed over DTLS-secured CoAP using any of the security modes described in Section 9 of RFC 7252. The use of DTLS is indicated by the "coaps" URI scheme. All notifications resulting from a GET request with an Observe Option MUST be returned within the same epoch of the same connection as the request.

8. IANA Considerations

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

```
+--------+---------+-----------+
| Number | Name    | Reference |
+--------+---------+-----------+
|       6| Observe | [RFCXXXX] |
+--------+---------+-----------+
```

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

9. Acknowledgements

Carsten Bormann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Daniele Alessandrelli, Jari Arkko, Peter A. Bigot, Angelo P. Castellani, Gilbert Clark, Esko Dijk, Thomas Fossati, Brian Frank, Bert Greevenbosch, Jeroen Hoebeke, Cullen Jennings, Matthias Kovatsch, Barry Leiba, Salvatore Loreto, Charles Palmer, Akbar Rahman, Zach Shelby, and Floris Van den Abeele for helpful comments and discussions that have shaped the document.

This work was supported in part by Klaus Tschira Foundation, Intel, Cisco, and Nokia.

10. References
10.1. Normative References


10.2. Informative References


Appendix A. Examples

A.1. Client/Server Examples

<table>
<thead>
<tr>
<th>Observed</th>
<th>CLIENT State</th>
<th>SERVER State</th>
<th>Actual State</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>unknown</td>
<td></td>
<td>18.5 Cel</td>
</tr>
<tr>
<td>3</td>
<td>+-----+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>GET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>&lt;-----+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>2.05</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>18.5 Cel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>&lt;-----+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>2.05</td>
<td>19.2 Cel</td>
</tr>
<tr>
<td>18</td>
<td>19.2 Cel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change.
<table>
<thead>
<tr>
<th>Observed t</th>
<th>CLIENT State</th>
<th>SERVER State</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>19.2 Cel</td>
<td></td>
<td>19.2 Cel</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>X----+</td>
<td>Header: 2.05 0x51457b51</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>2.05</td>
<td>19.7 Cel</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td>Observe: 25</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td>Max-Age: 15</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td>Payload: &quot;19.7 Cel&quot;</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>19.2 Cel</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>(stale)</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>+------&gt;</td>
<td></td>
<td>Header: GET 0x41011634</td>
</tr>
<tr>
<td>39</td>
<td>GET</td>
<td></td>
<td>Token: 0xb2</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>Uri-Path: temperature</td>
<td>Observe: 0 (register)</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>&lt;------</td>
<td></td>
<td>Header: 2.05 0x61451634</td>
</tr>
<tr>
<td>45</td>
<td>2.05</td>
<td></td>
<td>Token: 0xb2</td>
</tr>
<tr>
<td>46</td>
<td>19.7 Cel</td>
<td></td>
<td>Observe: 44</td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
<td>Max-Age: 15</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td>ETag: 0x78797a7a79</td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
<td>Payload: &quot;19.7 Cel&quot;</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: The client re-registers after Max-Age ends
<table>
<thead>
<tr>
<th>Observed t</th>
<th>CLIENT State</th>
<th>SERVER State</th>
<th>Actual State</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>19.7 Cel</td>
<td>19.7 Cel</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td>crash</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>19.7 Cel</td>
<td>20.0 Cel</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>(stale)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td></td>
<td>reboot</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
56  +-----> Header: GET 0x41011635
57   GET     Token: 0xf9
58   Uri-Path: temperature
59   Observe: 0 (register)
60   ETag: 0x78797a7a79
```

```
74  +----> Header: 2.05 0x61451635
75   2.05     Token: 0xf9
76   20.0 Cel Observe: 74
77              Max-Age: 15
78              Payload: "20.0 Cel"
```

```
81  +----> Header: 2.03 0x5143aa0c
82   2.03     Token: 0xf9
83   19.7 Cel Observe: 81
84              ETag: 0x78797a7a79
85              Max-Age: 15
```

Figure 5: The client re-registers and gives the server the opportunity to select a stored response.
<table>
<thead>
<tr>
<th>t</th>
<th>State</th>
<th>CLIENT</th>
<th>SERVER</th>
<th>Actual</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>19.7 Cel</td>
<td></td>
<td></td>
<td>19.7 Cel</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td>&lt;-----+</td>
<td>19.3 Cel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td></td>
<td>2.05</td>
<td></td>
<td>Observe: 91</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td>Max-Age: 15</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>19.3 Cel</td>
<td></td>
<td></td>
<td>Payload: &quot;19.3 Cel&quot;</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td></td>
<td>++ - -&gt;</td>
<td></td>
<td>Header: 0x7000aa0f</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
<td></td>
<td></td>
<td>19.0 Cel</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>19.3 Cel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>(stale)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: The client rejects a notification and thereby cancels the observation.
### A.2. Proxy Examples

<table>
<thead>
<tr>
<th>CLIENT</th>
<th>PROXY</th>
<th>SERVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>+------&gt;</td>
<td>Header: GET 0x41015fb8</td>
<td>Token: 0x1a</td>
</tr>
<tr>
<td>GET</td>
<td>Uri-Host: sensor.example</td>
<td>Uri-Path: status</td>
</tr>
<tr>
<td></td>
<td>Observe: 0 (register)</td>
<td></td>
</tr>
<tr>
<td>&lt;-</td>
<td>Header: 2.05 0x61455fb8</td>
<td>Token: 0x1a</td>
</tr>
<tr>
<td>2.05</td>
<td>Observe: 42</td>
<td>Max-Age: 60</td>
</tr>
<tr>
<td></td>
<td>Payload: &quot;ready&quot;</td>
<td></td>
</tr>
<tr>
<td>+------&gt;</td>
<td>Header: GET 0x41011633</td>
<td>Token: 0x9a</td>
</tr>
<tr>
<td>GET</td>
<td>Proxy-Uri: coap://sensor.example/status</td>
<td></td>
</tr>
<tr>
<td>&lt;-</td>
<td>Header: 2.05 0x61451633</td>
<td>Token: 0x9a</td>
</tr>
<tr>
<td>2.05</td>
<td>Max-Age: 53</td>
<td>Payload: &quot;ready&quot;</td>
</tr>
<tr>
<td>&lt;-</td>
<td>Header: 2.05 0x514505fc0</td>
<td>Token: 0x1a</td>
</tr>
<tr>
<td>2.05</td>
<td>Observe: 135</td>
<td>Max-Age: 60</td>
</tr>
<tr>
<td></td>
<td>Payload: &quot;busy&quot;</td>
<td></td>
</tr>
<tr>
<td>+------&gt;</td>
<td>Header: GET 0x41011634</td>
<td>Token: 0x9b</td>
</tr>
<tr>
<td>GET</td>
<td>Proxy-Uri: coap://sensor.example/status</td>
<td></td>
</tr>
<tr>
<td>&lt;-</td>
<td>Header: 2.05 0x61451634</td>
<td>Token: 0x9b</td>
</tr>
<tr>
<td>2.05</td>
<td>Max-Age: 49</td>
<td>Payload: &quot;busy&quot;</td>
</tr>
</tbody>
</table>

Figure 7: A proxy observes a resource to keep its cache up to date
Figure 8: A client observes a resource through a proxy

---

Internet-Draft Observing Resources in CoAP October 2014

---

Hartke Expires April 30, 2015 [Page 27]
Appendix B. Changelog

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

Changes from ietf-14 to ietf-15:

- Clarified several points based on AD, GenART, IESG, and Secdir reviews.

Changes from ietf-13 to ietf-14:

- Updated references.

Changes from ietf-12 to ietf-13:

- Extended the Observe Option in requests to not only add but also remove an entry in the list of observers, depending on the option value.

  Note: The value of the Observe Option in a registration request may now be any sequence of bytes that encodes the unsigned integer 0, i.e., 0x'', 0x'00'', 0x'00 00' or 0x'00 00 00'.

- Removed the 7.31 Code for cancellation.

Changes from ietf-11 to ietf-12:

- Introduced the 7.31 Code to request the cancellation of a pending request.

- Made the algorithm for superseding an outstanding transmission OPTIONAL.

- Clarified that the entry in the list of observers is removed if the client fails to acknowledge a confirmable notification before the last retransmission attempt times out (#350).

- Simplified the text on cancellation (#352) and the handling of Reset messages (#353).

Changes from ietf-10 to ietf-11:

- Pointed out that client and server clocks may differ in their realization of the SI second, and added robustness to the existing reordering scheme by reducing the maximum notification rate to 32768 notifications per second (#341).

Changes from ietf-09 to ietf-10:
- Required consistent sequence numbers across requests (#333).

- Clarified that a server needs to update the entry in the list of observers instead of adding a new entry if the endpoint/token pair is already present.

- Allowed that a client uses a token that is currently in use to ensure that it’s still in the list of observers. This is possible because sequence numbers are now consistent across requests and servers won’t add a new entry for the same token.

- Improved text on the transmission of non-confirable notifications to match Section 3.1.2 of RFC 5405 more closely.

- Updated examples to use UCUM units.

- Moved Appendix B into the introduction.

Changes from ietf-08 to ietf-09:

- Removed the side effects of requests on existing observations. This includes removing that
  * the client can use a GET request to cancel an observation;
  * the server updates the entry in the list of observers instead of adding a new entry if the client is already present (#258, #281).

- Clarified that a resource (and hence an observation relationship) is identified by the request options that are part of the Cache-Key (#258).

- Clarified that a non-2.xx notification MUST NOT include an Observe Option.

- Moved block-wise transfer of notifications to [I-D.ietf-core-block].

Changes from ietf-07 to ietf-08:

- Expanded text on transmitting a notification while a previous transmission is pending (#242).

- Changed reordering detection to use a fixed time span of 128 seconds instead of EXCHANGE_LIFETIME (#276).
- Removed the use of the freshness model to determine if the client is still on the list of observers. This includes removing that
  - the client assumes that it has been removed from the list of observers when Max-Age ends;
  - the server sets the Max-Age Option of a notification to a value that indicates when the server will send the next notification;
  - the server uses a number of retransmit attempts such that removing a client from the list of observers before Max-Age ends is avoided (#235);
  - the server may remove the client from all lists of observers when the transmission of a confirmable notification ultimately times out.

- Changed that an unrecognized critical option in a request must actually have no effect on the state of any observation relationship to any resource, as the option could lead to a different target resource.

- Clarified that client implementations must be prepared to receive each notification equally as a confirmable or a non-confirmable message, regardless of the message type of the request and of any previous notification.

- Added a requirement for sending a confirmable notification at least every 24 hours before continuing with non-confirmable notifications (#221).

- Added congestion control considerations from [I-D.bormann-core-congestion-control-02].

- Recommended that the client waits for a randomized time after the freshness of the latest notification expired before re-registering. This prevents that multiple clients observing a resource perform a GET request at the same time when the need to re-register arises.

- Changed reordering detection from 'MAY' to 'SHOULD', as the goal of the protocol (to keep the observed state as closely in sync with the actual state as possible) is not optional.

- Fixed the length of the Observe Option (3 bytes) in the table in Section 2.
o Replaced the ‘x’ in the No-Cache-Key column in the table in Section 2 with a ‘-’, as the Observe Option doesn’t have the No-Cache-Key flag set, even though it is not part of the cache key.

o Updated examples.

Changes from ietf-06 to ietf-07:

o Moved to 24-bit sequence numbers to allow for up to 15000 notifications per second per client and resource (#217).

o Re-numbered option number to use Unsafe/Safe and Cache-Key compliant numbers (#241).

o Clarified how to react to a Reset message that is sent in reply to a non-confirmable notification (#225).

o Clarified the semantics of the "obs" link target attribute (#236).

Changes from ietf-05 to ietf-06:

o Improved abstract and introduction to say that the protocol is about best effort and eventual consistency (#219).

o Clarified that the value of the Observe Option in a request must have zero length.

o Added requirement that the sequence number must be updated each time a server retransmits a notification.

o Clarified that a server must remove a client from the list of observers when it receives a GET request with an unrecognized critical option.

o Updated the text to use the endpoint concept from [I-D.ietf-core-coap] (#224).

o Improved the reordering text (#223).

Changes from ietf-04 to ietf-05:

o Recommended that a client does not re-register while a new notification from the server is still likely to arrive. This is to avoid that the request of the client and the last notification after max-age cross over each other (#174).

o Relaxed requirements when sending a Reset message in reply to non-confirmable notifications.
o Added an implementation note about careless GET requests (#184).
o Updated examples.

Changes from ietf-03 to ietf-04:
o Removed the "Max-OFE" Option.
o Allowed a Reset message in reply to non-confirmable notifications.
o Added a section on cancellation.
o Updated examples.

Changes from ietf-02 to ietf-03:
o Separated client-side and server-side requirements.
o Fixed uncertainty if client is still on the list of observers by introducing a liveliness model based on Max-Age and a new option called "Max-OFE" (#174).
o Simplified the text on message reordering (#129).
o Clarified requirements for intermediaries.
o Clarified the combination of blockwise transfers with notifications (#172).
o Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.
o Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:
o Removed the requirement of periodic refreshing (#126).
o The new "Observe" Option replaces the "Lifetime" Option.
o Introduced a new mechanism to detect message reordering.
o Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:
o Changed terminology from "subscriptions" to "observation relationships" (#33).
- Changed the name of the option to "Lifetime".
- Clarified establishment of observation relationships.
- Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).
- Clarified rules for establishing observation relationships (#68).
- Clarified conditions under which an observation relationship is terminated.
- Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).
- Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).
- Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).
- Added a mechanism to detect message reordering (#35).
- Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).
- Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).
- Described interaction with blockwise transfers (#36).
- Added Resource Discovery section (#99).
- Added IANA Considerations.
- Added Security Considerations (#40).
- Added examples (#38).
Author’s Address

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
EMail: hartke@tzi.org
Observing Resources in CoAP
draft-ietf-core-observe-16

Abstract

The Constrained Application Protocol (CoAP) is a RESTful application protocol for constrained nodes and networks. The state of a resource on a CoAP server can change over time. This document specifies a simple protocol extension for CoAP that enables CoAP clients to "observe" resources, i.e., to retrieve a representation of a resource and keep this representation updated by the server over a period of time. The protocol follows a best-effort approach for sending new representations to clients and provides eventual consistency between the state observed by each client and the actual resource state at the server.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 3, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ........................................... 3
   1.1. Background ....................................... 3
   1.2. Protocol Overview ................................ 3
   1.3. Consistency Model ................................ 5
   1.4. Observable Resources ............................. 6
   1.5. Requirements Notation ............................ 7
2. The Observe Option .................................... 7
3. Client-side Requirements ............................. 8
   3.1. Request .......................................... 8
   3.2. Notifications .................................... 8
   3.3. Caching ......................................... 9
   3.4. Reordering ...................................... 10
   3.5. Transmission ..................................... 11
   3.6. Cancellation .................................... 11
4. Server-side Requirements ............................. 12
   4.1. Request .......................................... 12
   4.2. Notifications .................................... 13
   4.3. Caching ......................................... 13
   4.4. Reordering ...................................... 14
   4.5. Transmission ..................................... 15
5. Intermediaries ........................................ 18
6. Web Linking ........................................... 19
7. Security Considerations ............................. 19
8. IANA Considerations ................................ 20
9. Acknowledgements .................................... 20
10. References ........................................... 21
    10.1. Normative References .......................... 21
    10.2. Informative References ........................ 21
Appendix A. Examples .................................... 22
    A.1. Client/Server Examples .......................... 22
    A.2. Proxy Examples .................................. 26
Appendix B. Changelog .................................. 28
1. Introduction

1.1. Background

The Constrained Application Protocol (CoAP) [RFC7252] is intended to provide RESTful services [REST] 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 [RFC7228].

The model of REST is that of a client exchanging representations of resources with a server, where a representation captures the current or intended state of a resource. The server is the authority for representations of the resources in its namespace. A client interested in the state of a resource initiates a request to the server, the server then returns a response with a representation of the resource that is current at the time of the request.

This model does not work well when a client is interested in having a current representation of a resource over a period of time. Existing approaches from HTTP, such as repeated polling or HTTP long polling [RFC6202], generate significant complexity and/or overhead and thus are less applicable in a constrained environment.

The protocol specified in this document extends the CoAP core protocol with a mechanism for a CoAP client to "observe" a resource on a CoAP server: the client retrieves a representation of the resource and requests this representation be updated by the server as long as the client is interested in the resource.

The protocol keeps the architectural properties of REST. It enables high scalability and efficiency through the support of caches and proxies. There is no intention, though, to solve the full set of problems that the existing HTTP solutions solve, or to replace publish/subscribe networks that solve a much more general problem [RFC5989].

1.2. Protocol Overview

The protocol is based on the well-known observer design pattern [GOF]. In this design pattern, components called "observers" register at a specific, known provider called the "subject" that they are interested in being notified whenever the subject undergoes a change in state. The subject is responsible for administering its list of registered observers. If multiple subjects are of interest to an observer, the observer must register separately for all of them.
The observer design pattern is realized in CoAP as follows:

Subject: In the context of CoAP, the subject is a resource in the namespace of a CoAP server. The state of the resource can change over time, ranging from infrequent updates to continuous state transformations.

Observer: An observer is a CoAP client that is interested in having a current representation of the resource at any given time.

Registration: A client registers its interest in a resource by initiating an extended GET request to the server. In addition to returning a representation of the target resource, this request causes the server to add the client to the list of observers of the resource.

Notification: Whenever the state of a resource changes, the server notifies each client in the list of observers of the resource. Each notification is an additional CoAP response sent by the server in reply to the single extended GET request, and includes a complete, updated representation of the new resource state.

Figure 2 below shows an example of a CoAP client registering its interest in a resource and receiving three notifications: the first with the current state upon registration, and then two upon changes to the resource state. Both the registration request and the notifications are identified as such by the presence of the Observe Option defined in this document. In notifications, the Observe Option additionally provides a sequence number for reordering detection. All notifications carry the token specified by the client, so the client can easily correlate them to the request.
A client remains on the list of observers as long as the server can determine the client’s continued interest in the resource. The server may send a notification in a confirmable CoAP message to request an acknowledgement from the client. When the client deregisters, rejects a notification, or the transmission of a notification times out after several transmission attempts, the client is considered no longer interested in the resource and is removed by the server from the list of observers.

1.3. Consistency Model

While a client is in the list of observers of a resource, the goal of the protocol is to keep the resource state observed by the client as closely in sync with the actual state at the server as possible.

It cannot be avoided that the client and the server become out of sync at times: First, there is always some latency between the change of the resource state and the receipt of the notification. Second, CoAP messages with notifications can get lost, which will cause the client to assume an old state until it receives a new notification.
And third, the server may erroneously come to the conclusion that the client is no longer interested in the resource, which will cause the server to stop sending notifications and the client to assume an old state until it eventually registers its interest again.

The protocol addresses this issue as follows:

- It follows a best-effort approach for sending the current representation to the client after a state change: Clients should see the new state after a state change as soon as possible, and they should see as many states as possible. This is limited by congestion control, however, so a client cannot rely on observing every single state that a resource might go through.

- It labels notifications with a maximum duration up to which it is acceptable for the observed state and the actual state to be out of sync. When the age of the notification received reaches this limit, the client cannot use the enclosed representation until it receives a new notification.

- It is designed on the principle of eventual consistency: The protocol guarantees that, if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state.

1.4. Observable Resources

A CoAP server is the authority for determining under what conditions resources change their state and thus when observers are notified of new resource states. The protocol does not offer explicit means for setting up triggers or thresholds; it is up to the server to expose observable resources that change their state in a way that is useful in the application context.

For example, a CoAP server with an attached temperature sensor could expose one or more of the following resources:

- `<coap://server/temperature>`, which changes its state every few seconds to a current reading of the temperature sensor;

- `<coap://server/temperature/felt>`, which changes its state to "COLD" whenever the temperature reading drops below a certain pre-configured threshold, and to "WARM" whenever the reading exceeds a second, slightly higher threshold;

- `<coap://server/temperature/critical?above=42>`, which changes its state based on the client-specified parameter value: every few seconds to the current temperature reading if the temperature
Thus, by designing CoAP resources that change their state on certain conditions, it is possible to update the client only when these conditions occur instead of supplying it continuously with raw sensor data. By parameterizing resources, this is not limited to conditions defined by the server, but can be extended to arbitrarily complex queries specified by the client. The application designer therefore can choose exactly the right level of complexity for the application envisioned and devices involved, and is not constrained to a "one size fits all" mechanism built into the protocol.

1.5. Requirements 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 RFC 2119 [RFC2119].

2. The Observe Option

The Observe Option has the following properties. Its meaning depends on whether it is included in a GET request or in a response.

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td></td>
<td>Observe</td>
<td>uint</td>
<td>0-3 B</td>
<td>(none)</td>
</tr>
</tbody>
</table>

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Table 1: The Observe Option

When included in a GET request, the Observe Option extends the GET method so it does not only retrieve a current representation of the target resource, but also requests the server to add or remove an entry in the list of observers of the resource, depending on the option value. The list entry consists of the client endpoint and the token specified by the client in the request. Possible values are:

0 (register) adds the entry to the list, if not present;

1 (deregister) removes the entry from the list, if present.
The Observe Option is not critical for processing the request. If the server is unwilling or unable to add a new entry to the list of observers, then the request falls back to a normal GET request, and the response does not include the Observe Option.

The Observe Option is not part of the cache-key: a cacheable response obtained with an Observe Option in the request can be used to satisfy a request without an Observe Option, and vice versa. When a stored response with an Observe Option is used to satisfy a normal GET request, the option MUST be removed before the response is returned.

When included in a response, the Observe Option identifies the message as a notification. This implies that a matching entry exists in the list of observers and that the server will notify the client of changes to the resource state. The option value is a sequence number for reordering detection (see Section 3.4 and Section 4.4).

The value of the Observe Option is encoded as an unsigned integer in network byte order using a variable number of bytes (‘uint’ option format); see Section 3.2 of RFC 7252 [RFC7252].

3. Client-side Requirements

3.1. Request

A client registers its interest in a resource by issuing a GET request with an Observe Option set to 0 (register). If the server returns a 2.xx response that includes an Observe Option as well, the server has successfully added an entry with the client endpoint and request token to the list of observers of the target resource and the client will be notified of changes to the resource state.

Like a fresh response can be used to satisfy a request without contacting the server, the stream of updates resulting from one observation request can be used to satisfy another (observation or normal GET) request if the target resource is the same. A client MUST aggregate such requests and MUST NOT register more than once for the same target resource. The target resource is identified by all options in the request that are part of the cache-key. This includes, for example, the full request URI and the Accept Option.

3.2. Notifications

Notifications are additional responses sent by the server in reply to the single extended GET request that created the registration. Each notification includes the token specified by the client in the request. The only difference between a notification and a normal response is the presence of the Observe Option.
Notifications typically have a 2.05 (Content) response code. They include an Observe Option with a sequence number for reordering detection (see Section 3.4), and a payload in the same Content-Format as the initial response. If the client included one or more ETag Options in the GET request (see Section 3.3), notifications can have a 2.03 (Valid) response code rather than a 2.05 (Content) response code. Such notifications include an Observe Option with a sequence number but no payload.

In the event that the resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server sends a notification with an appropriate response code (such as 4.04 Not Found) and removes the client’s entry from the list of observers of the resource. Non-2.xx responses do not include an Observe Option.

3.3. Caching

As notifications are just additional responses to a GET request, notifications partake in caching as defined in Section 5.6 of RFC 7252 [RFC7252]. Both the freshness model and the validation model are supported.

3.3.1. Freshness

A client MAY store a notification like a response in its cache and use a stored notification that is fresh without contacting the server. Like a response, a notification is considered fresh while its age is not greater than the value indicated by the Max-Age Option (and no newer notification/response has been received).

The server will do its best to keep the resource state observed by the client as closely in sync with the actual state as possible. However, a client cannot rely on observing every single state that a resource might go through. For example, if the network is congested or the state changes more frequently than the network can handle, the server can skip notifications for any number of intermediate states.

The server uses the Max-Age Option to indicate an age up to which it is acceptable that the observed state and the actual state are inconsistent. If the age of the latest notification becomes greater than its indicated Max-Age, then the client MUST NOT assume that the enclosed representation reflects the actual resource state.

To make sure it has a current representation and/or to re-register its interest in a resource, a client MAY issue a new GET request with the same token as the original at any time. All options MUST be identical to those in the original request, except for the set of
ETag Options. It is RECOMMENDED that the client does not issue the request while it still has a fresh notification/response for the resource in its cache. Additionally, the client SHOULD at least wait for a random amount of time between 5 and 15 seconds after Max-Age expired to reduce collisions with other clients.

3.3.2. Validation

When a client has one or more notifications stored in its cache for a resource, it can use the ETag Option in the GET request to give the server an opportunity to select a stored notification to be used.

The client MAY include an ETag Option for each stored response that is applicable in the GET request. Whenever the observed resource changes to a representation identified by one of the ETag Options, the server can select a stored response by sending a 2.03 (Valid) notification with an appropriate ETag Option instead of a 2.05 (Content) notification.

A client implementation needs to keep all candidate responses in its cache until it is no longer interested in the target resource or it re-registers with a new set of entity-tags.

3.4. Reordering

Messages with notifications can arrive in a different order than they were sent. Since the goal is to keep the observed state as closely in sync with the actual state as possible, a client MUST consider the notification that was sent most recently as the freshest, regardless of the order of arrival.

To provide an order among notifications for the client, the server sets the value of the Observe Option in each notification to the 24 least-significant bits of a strictly increasing sequence number. An incoming notification was sent more recently than the freshest notification so far when one of the following conditions is met:

\[
(V1 < V2 \text{ and } V2 - V1 < 2^{23}) \text{ or } \\
(V1 > V2 \text{ and } V1 - V2 > 2^{23}) \text{ or } \\
(T2 > T1 + 128 \text{ seconds})
\]

where \(V1\) is the value of the Observe Option in the freshest notification so far, \(V2\) the value of the Observe Option in the incoming notification, \(T1\) a client-local timestamp for the freshest notification so far, and \(T2\) a client-local timestamp for the incoming notification.
Design Note: The first two conditions verify that \( V_1 \) is less than \( V_2 \) in 24-bit serial number arithmetic [RFC1982]. The third condition ensures that, if the server is generating serial numbers based on a local clock, the time elapsed between the two incoming messages is not so large that the difference between \( V_1 \) and \( V_2 \) has become larger than the largest integer that it is meaningful to add to a 24-bit serial number; in other words, after 128 seconds have elapsed without any notification, a client does not need to check the sequence numbers to assume that an incoming notification was sent more recently than the freshest notification it has received so far.

The duration of 128 seconds was chosen as a nice round number greater than MAX_LATENCY (Section 4.8.2 of RFC 7252 [RFC7252]).

3.5. Transmission

A notification can be confirmable or non-confirmable, i.e., it can be sent in a confirmable or a non-confirmable message. The message type used for a notification is independent of the type used for the request and of any previous notification.

If a client does not recognize the token in a confirmable notification, it MUST NOT acknowledge the message and SHOULD reject it with a Reset message; otherwise, the client MUST acknowledge the message as usual. In the case of a non-confirmable notification, rejecting the message with a Reset message is OPTIONAL.

An acknowledgement message signals to the server that the client is alive and interested in receiving further notifications; if the server does not receive an acknowledgement in reply to a confirmable notification, it will assume that the client is no longer interested and will eventually remove the associated entry from the list of observers.

3.6. Cancellation

A client that is no longer interested in receiving notifications for a resource can simply "forget" the observation. When the server then sends the next notification, the client will not recognize the token in the message and thus will return a Reset message. This causes the server to remove the associated entry from the list of observers. The entries in lists of observers are effectively "garbage collected" by the server.
Implementation Note: Due to potential message loss, the Reset message may not reach the server. The client may therefore have to reject multiple notifications, each with one Reset message, until the server finally removes the associated entry from the list of observers and stops sending notifications.

In some circumstances, it may be desirable to cancel an observation and release the resources allocated by the server to it more eagerly. In this case, a client MAY explicitly deregister by issuing a GET request which has the Token field set to the token of the observation to be cancelled and includes an Observe Option with the value set to 1 (deregister). All other options MUST be identical to those in the registration request, except for the set of ETag Options. When the server receives such a request, it will remove any matching entry from the list of observers and process the GET request as usual.

4. Server-side Requirements

4.1. Request

A GET request with an Observe Option set to 0 (register) requests the server not only to return a current representation of the target resource, but also to add the client to the list of observers of that resource. Upon success, the server returns a current representation of the resource and MUST keep this representation updated (as described in Section 1.3) as long as the client is on the list of observers.

The entry in the list of observers is keyed by the client endpoint and the token specified by the client in the request. If an entry with a matching endpoint/token pair is already present in the list (which, for example, happens when the client wishes to reinforce its interest in a resource), the server MUST NOT add a new entry but MUST replace or update the existing one.

A server that is unable or unwilling to add a new entry to the list of observers of a resource MAY silently ignore the registration request and process the GET request as usual. The resulting response MUST NOT include an Observe Option, the absence of which signals to the client that it will not be notified of changes to the resource and, e.g., needs to poll the resource for its state instead.

If the Observe Option in a GET request is set to 1 (deregister), then the server MUST remove any existing entry with a matching endpoint/token pair from the list of observers and process the GET request as usual. The resulting response MUST NOT include an Observe Option.
4.2. Notifications

A client is notified of changes to the resource state by additional responses sent by the server in reply to the GET request. Each such notification response (including the initial response) MUST echo the token specified by the client in the GET request. If there are multiple entries in the list of observers, the order in which the clients are notified is not defined; the server is free to use any method to determine the order.

A notification SHOULD have a 2.05 (Content) or 2.03 (Valid) response code. However, in the event that the state of a resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server SHOULD notify the client by sending a notification with an appropriate response code (such as 4.04 Not Found) and subsequently MUST remove the associated entry from the list of observers of the resource.

The Content-Format specified in a 2.xx notification MUST be the same as the one used in the initial response to the GET request. If the server is unable to continue sending notifications in this format, it SHOULD send a notification with a 4.06 (Not Acceptable) response code and subsequently MUST remove the associated entry from the list of observers of the resource.

A 2.xx notification MUST include an Observe Option with a sequence number as specified in Section 4.4 below; a non-2.xx notification MUST NOT include an Observe Option.

4.3. Caching

As notifications are just additional responses sent by the server in reply to a GET request, they are subject to caching as defined in Section 5.6 of RFC 7252 [RFC7252].

4.3.1. Freshness

After returning the initial response, the server MUST keep the resource state that is observed by the client as closely in sync with the actual resource state as possible.

Since becoming out of sync at times cannot be avoided, the server MUST indicate for each representation an age up to which it is acceptable that the observed state and the actual state are inconsistent. This age is application-dependent and MUST be specified in notifications using the Max-Age Option.
When the resource does not change and the client has a current representation, the server does not need to send a notification. However, if the client does not receive a notification, the client cannot tell if the observed state and the actual state are still in sync. Thus, when the age of the latest notification becomes greater than its indicated Max-Age, the client no longer has a usable representation of the resource state. The server MAY wish to prevent that by sending a new notification with the unchanged representation and a new Max-Age just before the Max-Age indicated earlier expires.

4.3.2. Validation

A client can include a set of entity-tags in its request using the ETag Option. When a resource changes its state and the origin server is about to send a 2.05 (Content) notification, then, whenever that notification has an entity-tag in the set of entity-tags specified by the client, the server MAY send a 2.03 (Valid) response with an appropriate ETag Option instead.

4.4. Reordering

Because messages can get reordered, the client needs a way to determine if a notification arrived later than a newer notification. For this purpose, the server MUST set the value of the Observe Option of each notification it sends to the 24 least-significant bits of a strictly increasing sequence number. The sequence number MAY start at any value and MUST NOT increase so fast that it increases by more than $2^{23}$ within less than 256 seconds.

The sequence number selected for a notification MUST be greater than that of any preceding notification sent to the same client with the same token for the same resource. The value of the Observe Option MUST be current at the time of transmission; if a notification is retransmitted, the server MUST update the value of the option to the sequence number that is current at that time before retransmission.

Implementation Note: A simple implementation that satisfies the requirements is to obtain a timestamp from a local clock. The sequence number then is the timestamp in ticks, where 1 tick = $(256 \text{ seconds})/(2^{23}) = 30.52$ microseconds. It is not necessary that the clock reflects the current time/date.

Another valid implementation is to store a 24-bit unsigned integer variable per resource and increment this variable each time the resource undergoes a change of state (provided that the resource changes its state less than $2^{23}$ times in the first 256 seconds after every state change). This removes the need to update the value of the Observe Option on retransmission when the resource
state did not change.

Design Note: The choice of a 24-bit option value and a time span of 256 seconds theoretically allows for a notification rate of up to 65536 notifications per second. Constrained nodes often have rather imprecise clocks, though, and inaccuracies of the client and server side may cancel out or add in effect. Therefore, the maximum notification rate is reduced to 32768 notifications per second. This is still well beyond the highest known design objective of around 1 kHz (most CoAP applications will be several orders of magnitude below that), but allows total clock inaccuracies of up to -50/+100 %.

4.5. Transmission

A notification can be sent in a confirmable or a non-confirmable message. The message type used is typically application-dependent and may be determined by the server for each notification individually.

For example, for resources that change in a somewhat predictable or regular fashion, notifications can be sent in non-confirmable messages; for resources that change infrequently, notifications can be sent in confirmable messages. The server can combine these two approaches depending on the frequency of state changes and the importance of individual notifications.

A server MAY choose to skip sending a notification if it knows that it will send another notification soon, for example, when the state of a resource is changing frequently. It also MAY choose to send more than one notification for the same resource state. However, above all, the server MUST ensure that a client in the list of observers of a resource eventually observes the latest state if the resource does not undergo a new change in state.

For example, when state changes occur in bursts, the server can skip some notifications, send the notifications in non-confirmable messages, and make sure that the client observes the latest state change by repeating the last notification in a confirmable message when the burst is over.

The client’s acknowledgement of a confirmable notification signals that the client is interested in receiving further notifications. If a client rejects a confirmable or non-confirmable notification with a Reset message, or if the last attempt to retransmit a confirmable notification times out, then the client is considered no longer interested and the server MUST remove the associated entry from the list of observers.
Implementation Note: To properly process a Reset message that rejects a non-confirmable notification, a server needs to remember the message IDs of the non-confirmable notifications it sends. This may be challenging for a server with constrained resources. However, since Reset messages are transmitted unreliably, the client must be prepared that its Reset messages aren’t received by the server. A server thus can always pretend that a Reset message rejecting a non-confirmable notification was lost. If a server does this, it could accelerate cancellation by sending the following notifications to that client in confirmable messages.

A server that transmits notifications mostly in non-confirmable messages MUST send a notification in a confirmable message instead of a non-confirmable message at least every 24 hours. This prevents a client that went away or is no longer interested from remaining in the list of observers indefinitely.

4.5.1. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2 of RFC 7252 [RFC7252] and the limitations in Section 4.7 of RFC 7252 [RFC7252]. However, CoAP places the responsibility of congestion control for simple request/response interactions only on the clients: rate limiting request transmission implicitly controls the transmission of the responses. When a single request yields a potentially infinite number of notifications, additional responsibility needs to be placed on the server.

In order not to cause congestion, servers MUST strictly limit the number of simultaneous outstanding notifications/responses that they transmit to a given client to NSTART (1 by default; see Section 4.7 of RFC 7252 [RFC7252]). An outstanding notification/response is either a confirmable message for which an acknowledgement has not yet been received and whose last retransmission attempt has not yet timed out, or a non-confirmable message for which the waiting time that results from the following rate limiting rules has not yet elapsed.

The server SHOULD NOT send more than one non-confirmable notification per round-trip time (RTT) to a client on average. If the server cannot maintain an RTT estimate for a client, it SHOULD NOT send more than one non-confirmable notification every 3 seconds, and SHOULD use an even less aggressive rate when possible (see also Section 3.1.2 of RFC 5405 [RFC5405]).

Further congestion control optimizations and considerations are expected in the future with advanced CoAP congestion control mechanisms.
4.5.2. Advanced Transmission

The state of an observed resource may change while the number of the number of simultaneous outstanding notifications/responses to a client on the list of observers is greater than or equal to NSTART. In this case, the server cannot notify the client of the new resource state immediately but has to wait for an outstanding notification/response to complete first.

If there exists an outstanding notification/response that the server transmits to the client and that pertains to the changed resource, then it is desirable for the server to stop working towards getting the representation of the old resource state to the client, and to start transmitting the current representation to the client instead, so the resource state observed by the client stays closer in sync with the actual state at the server.

For this purpose, the server MAY optimize the transmission process by aborting the transmission of the old notification (but not before the current transmission attempt completed) and starting a new transmission for the new notification (but with the retransmission timer and counter of the aborted transmission retained).

In more detail, a server MAY supersede an outstanding transmission that pertains to an observation as follows:

1. Wait for the current (re-)transmission attempt to be acknowledged, rejected or to time out (confirmable transmission); or wait for the waiting time to elapse or the transmission to be rejected (non-confirmable transmission).

2. If the transmission is rejected or it was the last attempt to retransmit a notification, remove the associated entry from the list of observers of the observed resource.

3. If the entry is still in the list of observers, start to transmit a new notification with a representation of the current resource state. Should the resource have changed its state more than once in the meantime, the notifications for the intermediate states are silently skipped.

4. The new notification is transmitted with a new Message ID and the following transmission parameters: If the previous (re-)transmission attempt timed out, retain its transmission parameters, increment the retransmission counter and double the timeout; otherwise, initialize the transmission parameters as usual (see Section 4.2 of RFC 7252 [RFC7252]).
It is possible that the server later receives an acknowledgement for a confirmable notification that it superseded this way. Even though this does not signal consistency, it is valuable in that it signals the client’s further interest in the resource. The server therefore should avoid inadvertently removing the associated entry from the list of observers.

5. Intermediaries

A client may be interested in a resource in the namespace of a server that is reached through a chain of one or more CoAP intermediaries. In this case, the client registers its interest with the first intermediary towards the server, acting as if it was communicating with the server itself, as specified in Section 3. It is the task of this intermediary to provide the client with a current representation of the target resource and to keep the representation updated upon changes to the resource state, as specified in Section 4.

To perform this task, the intermediary SHOULD make use of the protocol specified in this document, taking the role of the client and registering its own interest in the target resource with the next hop towards the server. If the response returned by the next hop doesn’t include an Observe Option, the intermediary MAY resort to polling the next hop or MAY itself return a response without an Observe Option.

The communication between each pair of hops is independent; each hop in the server role MUST determine individually how many notifications to send, of which message type, and so on. Each hop MUST generate its own values for the Observe Option in notifications, and MUST set the value of the Max-Age Option according to the age of the local current representation.

If two or more clients have registered their interest in a resource with an intermediary, the intermediary MUST register itself only once with the next hop and fan out the notifications it receives to all registered clients. This relieves the next hop from sending the same notifications multiple times and thus enables scalability.

An intermediary is not required to act on behalf of a client to observe a resource; an intermediary MAY observe a resource, for example, just to keep its own cache up to date.

See Appendix A.2 for examples.
6. Web Linking

A web link [RFC5988] to a resource accessible over CoAP (for example, in a link-format document [RFC6690]) MAY include the target attribute "obs".

The "obs" attribute, when present, is a hint indicating that the destination of a link is useful for observation and thus, for example, should have a suitable graphical representation in a user interface. Note that this is only a hint; it is not a promise that the Observe Option can actually be used to perform the observation. A client may need to resort to polling the resource if the Observe Option is not returned in the response to the GET request.

A value MUST NOT be given for the "obs" attribute; any present value MUST be ignored by parsers. The "obs" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by parsers.

7. Security Considerations

The security considerations in Section 11 of the CoAP specification [RFC7252] apply.

Observing resources can dramatically increase the negative effects of amplification attacks. That is, not only can notifications messages be much larger than the request message, but the nature of the protocol can cause a significant number of notifications to be generated. Without client authentication, a server therefore MUST strictly limit the number of notifications that it sends between receiving acknowledgements that confirm the actual interest of the client in the data; i.e., any notifications sent in non-confirmable messages MUST be interspersed with confirmable messages. (An attacker may still spoof the acknowledgements if the confirmable messages are sufficiently predictable.)

The protocol follows a best-effort approach for keeping the state observed by a client and the actual resource state at a server in sync. This may have the client and the server become out of sync at times. Depending on the sensitivity of the observed resource, operating on an old state might be a security threat. The client therefore must be careful not to use a representation after its Max-Age expires, and the server must set the Max-Age Option to a sensible value.

As with any protocol that creates state, attackers may attempt to exhaust the resources that the server has available for maintaining the list of observers for each resource. Servers may want to apply
access controls to this creation of state. As degraded behavior, the server can always fall back to processing the request as a normal GET request (without an Observe Option) if it is unwilling or unable to add a client to the list of observers of a resource, including if system resources are exhausted or nearing exhaustion.

Intermediaries must be careful to ensure that notifications cannot be employed to create a loop. A simple way to break any loops is to employ caches for forwarding notifications in intermediaries.

Resources can be observed over DTLS-secured CoAP using any of the security modes described in Section 9 of RFC 7252. The use of DTLS is indicated by the "coaps" URI scheme. All notifications resulting from a GET request with an Observe Option MUST be returned within the same epoch of the same connection as the request.

8. IANA Considerations

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

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Observe</td>
<td>[RFCXXXX]</td>
</tr>
</tbody>
</table>

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

9. Acknowledgements

Carsten Bormann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Daniele Alessandrelli, Jari Arkko, Peter A. Bigot, Angelo P. Castellani, Gilbert Clark, Esko Dijk, Thomas Fossati, Brian Frank, Bert Greevenbosch, Jeroen Hoebeke, Cullen Jennings, Matthias Kovatsch, Barry Leiba, Salvatore Loreto, Charles Palmer, Akbar Rahman, Zach Shelby, and Floris Van den Abeele for helpful comments and discussions that have shaped the document.

This work was supported in part by Klaus Tschira Foundation, Intel, Cisco, and Nokia.

10. References
10.1.  Normative References


10.2.  Informative References


Appendix A. Examples

A.1. Client/Server Examples

<table>
<thead>
<tr>
<th>t</th>
<th>Observed</th>
<th>CLIENT</th>
<th>SERVER</th>
<th>Actual State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>unknown</td>
<td></td>
<td></td>
<td>18.5 Cel</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>Header: GET 0x41011633</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Token: 0x4a</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>GET</td>
<td></td>
<td>Uri-Path: temperature</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>Observe: 0 (register)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>__________</td>
<td>&lt;-----+</td>
<td></td>
<td>Header: 2.05 0x61451633</td>
</tr>
<tr>
<td>10</td>
<td>__________</td>
<td></td>
<td>2.05</td>
<td>Token: 0x4a</td>
</tr>
<tr>
<td>11</td>
<td>18.5 Cel</td>
<td></td>
<td></td>
<td>Observe: 9</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>Max-Age: 15</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>Payload: &quot;18.5 Cel&quot;</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>__________</td>
<td>&lt;-----+</td>
<td></td>
<td>Header: 2.05 0x51457b50</td>
</tr>
<tr>
<td>17</td>
<td>__________</td>
<td></td>
<td>2.05</td>
<td>Token: 0x4a</td>
</tr>
<tr>
<td>18</td>
<td>19.2 Cel</td>
<td></td>
<td></td>
<td>Observe: 16</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td>Max-Age: 15</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>Payload: &quot;19.2 Cel&quot;</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change
<table>
<thead>
<tr>
<th>t</th>
<th>Observed</th>
<th>CLIENT State</th>
<th>SERVER State</th>
<th>Actual State</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>19.2 Cel</td>
<td>19.2 Cel</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>X----+</td>
<td>2.05</td>
<td>Header: 2.05 0x51457b51</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>2.05</td>
<td>19.7 Cel</td>
<td>Token: 0x4a</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td>Observe: 25</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td>Max-Age: 15</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td>Payload: &quot;19.7 Cel&quot;</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>19.2 Cel</td>
<td>(stale)</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td>&lt;-----+</td>
<td>Header: GET 0x41011634</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
<td>GET</td>
<td>Token: 0xb2</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td>Uri-Path: temperature</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
<td>Observe: 0 (register)</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>2.05</td>
<td></td>
<td>Header: 2.05 0x61451634</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td>Token: 0xb2</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td>19.7 Cel</td>
<td>Observe: 44</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
<td>Max-Age: 15</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td>ETag: 0x78797a7a79</td>
<td>Payload: &quot;19.7 Cel&quot;</td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: The client re-registers after Max-Age ends
<table>
<thead>
<tr>
<th>Observed t</th>
<th>CLIENT State</th>
<th>SERVER State</th>
<th>Actual State</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>19.7 Cel</td>
<td></td>
<td>19.7 Cel</td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
<td>crash</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>19.7 Cel</td>
<td></td>
<td>20.0 Cel</td>
</tr>
<tr>
<td>61</td>
<td>(stale)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>++----&gt;</td>
<td></td>
<td>Header: GET 0x41011635</td>
</tr>
<tr>
<td>68</td>
<td>GET</td>
<td>Token: 0xf9</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td></td>
<td>Uri-Path: temperature</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>Observe: 0 (register)</td>
<td>ETag: 0x78797a7a79</td>
</tr>
<tr>
<td>71</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>&lt;-----&lt;</td>
<td></td>
<td>Header: 2.05 0x61451635</td>
</tr>
<tr>
<td>74</td>
<td>2.05</td>
<td>Token: 0xf9</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td></td>
<td>Observe: 74</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>20.0 Cel</td>
<td>Max-Age: 15</td>
<td>Payload: &quot;20.0 Cel&quot;</td>
</tr>
<tr>
<td>77</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>&lt;-----&lt;</td>
<td></td>
<td>Header: 2.03 0x5143aa0c</td>
</tr>
<tr>
<td>82</td>
<td>2.03</td>
<td>Token: 0xf9</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>19.7 Cel</td>
<td>Observe: 81</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td></td>
<td>ETag: 0x78797a7a79</td>
<td>Max-Age: 15</td>
</tr>
<tr>
<td>85</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: The client re-registers and gives the server the opportunity to select a stored response
<table>
<thead>
<tr>
<th>t</th>
<th>Observed State</th>
<th>CLIENT State</th>
<th>SERVER State</th>
<th>Actual State</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>19.7 Cel</td>
<td></td>
<td></td>
<td>19.7 Cel</td>
</tr>
<tr>
<td>89</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>19.3 Cel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
<td></td>
<td></td>
<td>19.0 Cel</td>
</tr>
<tr>
<td>105</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>19.3 Cel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>(stale)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: The client rejects a notification and thereby cancels the observation.
A.2. Proxy Examples

Figure 7: A proxy observes a resource to keep its cache up to date
Figure 8: A client observes a resource through a proxy
Appendix B. Changelog

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

Changes from ietf-15 to ietf-16:

- Clarified several points based on AD, GenART, IESG, and Secdir reviews.

Changes from ietf-14 to ietf-15:

- Clarified several points based on AD, GenART, IESG, and Secdir reviews.

Changes from ietf-13 to ietf-14:

- Updated references.

Changes from ietf-12 to ietf-13:

- Extended the Observe Option in requests to not only add but also remove an entry in the list of observers, depending on the option value.

  Note: The value of the Observe Option in a registration request may now be any sequence of bytes that encodes the unsigned integer 0, i.e., 0x'', 0x'00', 0x'00 00' or 0x'00 00 00'.

- Removed the 7.31 Code for cancellation.

Changes from ietf-11 to ietf-12:

- Introduced the 7.31 Code to request the cancellation of a pending request.

- Made the algorithm for superseding an outstanding transmission OPTIONAL.

- Clarified that the entry in the list of observers is removed if the client fails to acknowledge a confirmable notification before the last retransmission attempt times out (#350).

- Simplified the text on cancellation (#352) and the handling of Reset messages (#353).

Changes from ietf-10 to ietf-11:
- Pointed out that client and server clocks may differ in their realization of the SI second, and added robustness to the existing reordering scheme by reducing the maximum notification rate to 32768 notifications per second (#341).

Changes from ietf-09 to ietf-10:

- Required consistent sequence numbers across requests (#333).

- Clarified that a server needs to update the entry in the list of observers instead of adding a new entry if the endpoint/token pair is already present.

- Allowed that a client uses a token that is currently in use to ensure that it’s still in the list of observers. This is possible because sequence numbers are now consistent across requests and servers won’t add a new entry for the same token.

- Improved text on the transmission of non-confirmable notifications to match Section 3.1.2 of RFC 5405 more closely.

- Updated examples to use UCUM units.

- Moved Appendix B into the introduction.

Changes from ietf-08 to ietf-09:

- Removed the side effects of requests on existing observations. This includes removing that
  - the client can use a GET request to cancel an observation;
  - the server updates the entry in the list of observers instead of adding a new entry if the client is already present (#258, #281).

- Clarified that a resource (and hence an observation relationship) is identified by the request options that are part of the Cache-Key (#258).

- Clarified that a non-2.xx notification MUST NOT include an Observe Option.

- Moved block-wise transfer of notifications to [I-D.ietf-core-block].

Changes from ietf-07 to ietf-08:
- Expanded text on transmitting a notification while a previous transmission is pending (#242).
- Changed reordering detection to use a fixed time span of 128 seconds instead of EXCHANGE_LIFETIME (#276).
- Removed the use of the freshness model to determine if the client is still on the list of observers. This includes removing that
  * the client assumes that it has been removed from the list of observers when Max-Age ends;
  * the server sets the Max-Age Option of a notification to a value that indicates when the server will send the next notification;
  * the server uses a number of retransmit attempts such that removing a client from the list of observers before Max-Age ends is avoided (#235);
  * the server may remove the client from all lists of observers when the transmission of a confirmable notification ultimately times out.
- Changed that an unrecognized critical option in a request must actually have no effect on the state of any observation relationship to any resource, as the option could lead to a different target resource.
- Clarified that client implementations must be prepared to receive each notification equally as a confirmable or a non-confirmable message, regardless of the message type of the request and of any previous notification.
- Added a requirement for sending a confirmable notification at least every 24 hours before continuing with non-confirmable notifications (#221).
- Added congestion control considerations from [I-D.bormann-core-congestion-control-02].
- Recommended that the client waits for a randomized time after the freshness of the latest notification expired before re-registering. This prevents that multiple clients observing a resource perform a GET request at the same time when the need to re-register arises.
- Changed reordering detection from ‘MAY’ to ‘SHOULD’, as the goal of the protocol (to keep the observed state as closely in sync
with the actual state as possible) is not optional.

- Fixed the length of the Observe Option (3 bytes) in the table in Section 2.
- Replaced the ‘x’ in the No-Cache-Key column in the table in Section 2 with a ‘-‘, as the Observe Option doesn’t have the No-Cache-Key flag set, even though it is not part of the cache key.
- Updated examples.

Changes from ietf-06 to ietf-07:

- Moved to 24-bit sequence numbers to allow for up to 15000 notifications per second per client and resource (#217).
- Re-numbered option number to use Unsafe/Safe and Cache-Key compliant numbers (#241).
- Clarified how to react to a Reset message that is sent in reply to a non-confirmable notification (#225).
- Clarified the semantics of the "obs" link target attribute (#236).

Changes from ietf-05 to ietf-06:

- Improved abstract and introduction to say that the protocol is about best effort and eventual consistency (#219).
- Clarified that the value of the Observe Option in a request must have zero length.
- Added requirement that the sequence number must be updated each time a server retransmits a notification.
- Clarified that a server must remove a client from the list of observers when it receives a GET request with an unrecognized critical option.
- Updated the text to use the endpoint concept from [I-D.ietf-core-coap] (#224).
- Improved the reordering text (#223).

Changes from ietf-04 to ietf-05:

- Recommended that a client does not re-register while a new notification from the server is still likely to arrive. This is
to avoid that the request of the client and the last notification after max-age cross over each other (#174).

- Relaxed requirements when sending a Reset message in reply to non-confirmable notifications.
- Added an implementation note about careless GET requests (#184).
- Updated examples.

Changes from ietf-03 to ietf-04:

- Removed the "Max-OFE" Option.
- Allowed a Reset message in reply to non-confirmable notifications.
- Added a section on cancellation.
- Updated examples.

Changes from ietf-02 to ietf-03:

- Separated client-side and server-side requirements.
- Fixed uncertainty if client is still on the list of observers by introducing a liveliness model based on Max-Age and a new option called "Max-OFE" (#174).
- Simplified the text on message reordering (#129).
- Clarified requirements for intermediaries.
- Clarified the combination of blockwise transfers with notifications (#172).
- Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.
- Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:

- Removed the requirement of periodic refreshing (#126).
- The new "Observe" Option replaces the "Lifetime" Option.
- Introduced a new mechanism to detect message reordering.
o Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

o Changed terminology from "subscriptions" to "observation relationships" (#33).

o Changed the name of the option to "Lifetime".

o Clarified establishment of observation relationships.

o Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).

o Clarified rules for establishing observation relationships (#68).

o Clarified conditions under which an observation relationship is terminated.

o Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).

o Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).

o Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).

o Added a mechanism to detect message reordering (#35).

o Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).

o Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).

o Described interaction with blockwise transfers (#36).

o Added Resource Discovery section (#99).

o Added IANA Considerations.

o Added Security Considerations (#40).

o Added examples (#38).
Author’s Address

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen  D-28359
Germany

Phone: +49-421-218-63905
EMail: hartke@tzi.org
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 May 13, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document.
publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
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 do not have routable addresses. 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 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 or link-format+json 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.

- a non-empty link-format document, which indicates the specific services that the requesting server wants to make known to the directory server.

The directory server integrates the information it received this way into its resource directory. It MAY make the information available to further directories, if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

The following example shows an endpoint using simple resource discovery, by simply sending a POST with its links in the body to a directory.

```
EP                                               RD
|                                                 |
| -- POST /.well-known/core "<sen/temp>..." ---> |
|  <---- 2.01 Created ------------------------- |
```
4.1. Finding a Directory Server

Endpoints that want to contact a directory server can obtain candidate IP addresses for such servers in a number of ways.

In a 6LoWPAN, good candidates can be taken from:

- 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 POST requests to that well-known multicast address (details TBD).

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]. 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.

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)

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 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.

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] or JSON Link Format [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 (mandatory). The endpoint identifier or name of the registering node, unique within that 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

The following response codes are defined for this interface:

Success: 2.01 "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". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an endpoint with the name "node1" registering two resources to an RD using this interface. The resulting location /rd/4521 is just an example of an RD generated location.

EP                                                RD
--- POST /rd?ep=node1 "</sensors..." -------->
<-- 2.01 Created Location: /rd/4521 ----------
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)

The following response codes are defined for this interface:

Success: 2.04 "Changed" in the update was successfully processed.

Failure: 4.00 "Bad Request". Malformed request.

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

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an endpoint updating a new set of resources to an RD using this interface.

```
EP                                                RD
|                                                 |
| --- POST /rd/4521 -------------------------->   |
|                                                 |
|                                                 |
<-- 2.04 Changed ----------------------------  |

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" upon successful deletion

Failure: 4.00 "Bad Request". Malformed request.

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

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following examples shows successful removal of the endpoint from the RD.

EP                                                RD
|                                                 |
| --- DELETE /rd/4521  ------------------------>  |
|                                                 |
|                                                 |
| <-- 2.02 Deleted  ----------------------------  |
|                                                 |

Req: DELETE /rd/4521

Res: 2.02 Deleted
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 [I-D.ietf-core-groupcomm].

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

The following response codes are defined for this interface:

Success: 2.01 "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". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows a group with the name "lights" registering two endpoints to an RD using this interface. The resulting location /rd-group/12 is just an example of an RD generated group location.

```
EP                                                RD
|                                                 |
| - POST /rd-group?gp=lights "<>;ep=node1..." --> |
| <---- 2.01 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" upon successful deletion

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.04 "Not Found". Group does not exist.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following examples show successful removal of the group from the RD.

<table>
<thead>
<tr>
<th>EP</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE /rd-group/412</td>
<td>---</td>
</tr>
</tbody>
</table>
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 JSON form ("application/link-format+json"). 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}

Parameters:

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 (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 contain '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" with an "application/link-format" or "application/link-format+json" payload containing a matching entries for the lookup.

Failure:  4.04 "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". Malformed request.

Failure:  5.03 "Service Unavailable". Service could not perform the operation.

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

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

Res: 2.05 Content
<coap://[ip:port]>;ep="node5",
<coap://[ip:port]>;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:

<table>
<thead>
<tr>
<th>Client</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>----- GET /rd-lookup/gp -----------------------&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;-- 2.05 Content /rd-group/12;gp=&quot;lights1&quot;; -----------------   d=&quot;example.com&quot; -----------------</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Client</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>----- GET /rd-lookup/ep?gp=lights1-----------------------&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;-- 2.05 Content <a href="">coap://{host:port}</a>;ep=&quot;node1&quot; ----------</td>
<td></td>
</tr>
</tbody>
</table>
<coap://{host:port}>;ep="node2",

The following example shows a client performing a lookup for all groups an endpoint belongs to:

Client                                               RD
| ----- GET /rd-lookup/gp?ep=node1 ------------------------> |
|                                                          |
| <-- 2.05 Content <coap://{ip:port}>;gp="lights1";ep="node1" |

Req: GET /rd-lookup/gp?ep=node1

Res: 2.05 Content
<coap://{ip:port}>;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
```

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 are defined from the "d" attribute. The domain attribute is suffixed to the host name and should be consistent with the domain name attributed to the hosting network segment.

9.5. TXT Record key=value strings

A number of [RFC6763] key/value pairs are derived from link-format information, to be exported in the DNS-SD as key=value strings in a TXT record ([RFC6763], Section 6.3).

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

The DNS TXT record can be further populated by importing any other resource description attributes as they share the same key=value format specified in Section 6 of [RFC6763].

9.6. Importing resource links into DNS-SD

Assuming the ability to query a Resource Directory or multicast a GET (?exp) over the local link, CoAP resource discovery may be used to populate the DNS-SD database in an automated fashion. CoAP resource descriptions (links) can be exported to DNS-SD for exposure to service discovery by using the Resource Instance attribute as the basis for a unique service name, composed with the Resource Type as the <ServiceType>, and registered in the correct <Domain>. The agent responsible for exporting records to the DNS zone file SHOULD be authenticated to the DNS server. The following example shows an agent discovering a resource to be exported:

Agent
--- GET /rd-lookup/res?exp ------------------------------>

2.05 Content "<coap://node1/light/1>;exp;rt="dali.light";ins="FrontSpot" d="example.com"

Req: GET /rd-lookup/res?exp

Res: 2.05 Content
<coap://[FDFD::1234]:61616/light/1>;exp;ct=41;rt="dali.light";ins="FrontSpot";
   d="example.com"

The agent subsequently registers the following DNS-SD RRs:
nodel.example.com.                IN AAAA
                FDFD::1234
_dali._udp.example.com IN PTR
                FrontSpot._dali._udp.example.com
light._sub._dali._udp.example.com IN PTR
                FrontSpot._dali._udp.example.com
FrontSpot._dali._udp.example.com IN SRV  0 0 5678
                nodel.example.com.
FrontSpot._dali._udp.example.com IN TXT
                txtver=1;path=/light/1

In the above figure the Service Name is chosen as
FrontSpot._dali._udp.example.com without the light._sub service
prefix.  An alternative Service Name would be:
FrontSpot.light._sub._dali._udp.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 (TODO:
Improve the exact DTLS or TLS security requirements and references).

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
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) [TODO: Ref, and cut down the verbiage, as this is already discussed in RFC 7252] 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).

Shelby & Bormann          Expires May 13, 2015                 [Page 31]
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Query</th>
<th>Validity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint Name</td>
<td>ep</td>
<td></td>
<td>Name of the endpoint</td>
</tr>
<tr>
<td>Lifetime</td>
<td>lt</td>
<td>60-4294967295</td>
<td>Lifetime of the registration in seconds</td>
</tr>
<tr>
<td>Domain</td>
<td>d</td>
<td></td>
<td>Domain to which this endpoint belongs</td>
</tr>
<tr>
<td>Endpoint Type</td>
<td>et</td>
<td></td>
<td>Semantic name of the endpoint</td>
</tr>
<tr>
<td>Context</td>
<td>con</td>
<td>URI</td>
<td>The scheme, address and port at which this server is available</td>
</tr>
<tr>
<td>Endpoint Name</td>
<td>ep</td>
<td></td>
<td>Name of the endpoint, max 63 bytes</td>
</tr>
<tr>
<td>Group</td>
<td>gp</td>
<td></td>
<td>Name of a group in the RD</td>
</tr>
<tr>
<td>Page</td>
<td>page</td>
<td>Integer</td>
<td>Used for pagination</td>
</tr>
<tr>
<td>Count</td>
<td>count</td>
<td>Integer</td>
<td>Used for pagination</td>
</tr>
</tbody>
</table>

Table 1: RD Parameters

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC5226].

12. Examples
13. Acknowledgments

Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Peter van der Stok, Anders Brandt, Matthieu Vial, Michael Koster, 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 -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.

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.
o Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.

o Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

15. References

15.1. Normative References

[I-D.ietf-core-links-json]
Bormann, C., "Representing CoRE Link Collections in JSON", draft-ietf-core-links-json-02 (work in progress), July 2014.


15.2. Informative References


Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
FINLAND

Phone: +1-408-203-9434
Email: zach.shelby@arm.com
Carsten Bormann
Universitaet Bremen T2I
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org
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), eXtensible Markup Language (XML) and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use this media type in protocols such as HTTP or CoAP to transport the measurements of the sensor or to be configured.

Status of this Memo

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

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

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

This Internet-Draft will expire on May 16, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document.
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 [RFC7252] 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 (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 object (with attributes) that contains an array 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 [RFC4627], 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.

```json
{"e": [{ "n": "urn:dev:ow:10e2073a01080063", "v":23.5, "u":"Cel" }]
```

In the example above, the array in the object 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 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

4. Semantics

Each representation carries a single SenML object that represents a set of measurements and/or parameters. This object contains 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 1 if not present.
Measurement or Parameter Entries

Array of values for sensor measurements or other generic parameters (such as configuration parameters). If present there must be at least one entry in the array.

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.

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

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 the base object, or in an entry. Extensions in the base object pertain to all entries, whereas extensions in an entry object only pertain to that.

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 RFC which 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 is 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.
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)

Root variables:
<table>
<thead>
<tr>
<th>SenML</th>
<th>JSON</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Name</td>
<td>bn</td>
<td>String</td>
</tr>
<tr>
<td>Base Time</td>
<td>bt</td>
<td>Number</td>
</tr>
<tr>
<td>Base Units</td>
<td>bu</td>
<td>Number</td>
</tr>
<tr>
<td>Version</td>
<td>ver</td>
<td>Number</td>
</tr>
<tr>
<td>Measurement or Parameters</td>
<td>e</td>
<td>Array</td>
</tr>
</tbody>
</table>

**Measurement or Parameter Entries:**

<table>
<thead>
<tr>
<th>SenML</th>
<th>JSON</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>n</td>
<td>String</td>
</tr>
<tr>
<td>Units</td>
<td>u</td>
<td>String</td>
</tr>
<tr>
<td>Value</td>
<td>v</td>
<td>Floating point</td>
</tr>
<tr>
<td>String Value</td>
<td>sv</td>
<td>String</td>
</tr>
<tr>
<td>Boolean Value</td>
<td>bv</td>
<td>Boolean</td>
</tr>
<tr>
<td>Value Sum</td>
<td>s</td>
<td>Floating point</td>
</tr>
<tr>
<td>Time</td>
<td>t</td>
<td>Number</td>
</tr>
<tr>
<td>Update Time</td>
<td>ut</td>
<td>Number</td>
</tr>
</tbody>
</table>

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 contents MUST consist of exactly one JSON object as specified by [RFC4627]. This object MAY contain a "bn" attribute with a value of type string. This 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, and the object MUST contain exactly one "e" attribute with a value of type array. The array MUST have one or more measurement or parameter objects.

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 microsecond 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:

{"e": [{ "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.

{"e": [
{ "n": "voltage", "t": 0, "u": "V", "v": 120.1 },
{ "n": "current", "t": 0, "u": "A", "v": 1.2 }
],
"bn": "urn:dev:mac:0024befffe804ff1/"
}

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.

{"e": [
{ "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 }
],
"bn": "urn:dev:mac:0024befffe804ff1/",
"bt": 1276020076,
"ver": 1,
"bu": "A"}
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 provide 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.

```json
{"e": [
    { "v": 20.0, "t": 0 },
    { "sv": "E 24° 30.621", "u": "lon", "t": 0 },
    { "sv": "N 60° 7.965", "u": "lat", "t": 0 },
    { "v": 20.3, "t": 60 },
    { "sv": "E 24° 30.622", "u": "lon", "t": 60 },
    { "sv": "N 60° 7.965", "u": "lat", "t": 60 },
    { "v": 20.7, "t": 120 },
    { "sv": "E 24° 30.623", "u": "lon", "t": 120 },
    { "sv": "N 60° 7.966", "u": "lat", "t": 120 },
    { "v": 98.0, "u": "%EL", "t": 150 },
    { "v": 21.3, "t": 180 },
    { "sv": "E 24° 30.628", "u": "lon", "t": 180 },
    { "sv": "N 60° 7.967", "u": "lat", "t": 180 }
],
"bn": "http://[2001:db8::1]",
"bt": 1320067464,
"bu": "%RH"
}
```

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.

```json
{"e": [
    { "n": "temperature", "v": 27.2, "u": "Cel" },
    { "n": "humidity", "v": 80, "u": "%RH" }
],
"bn": "http://[2001:db8::2]",
"bt": 1320067464,
"bu": "%RH"
}
```
7. 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
<?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

8. 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 as 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.
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
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 make 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 XML SenML file such as:

```xml
<?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="degC" />
</senml>
```

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

```
00000000  a00048806c200200 1d75726e3a646576 |..H.l ...urn:dev|
00000010  3a6f773a1306532 3037336130313038 |:ow:10e2073a0108|
00000020  3030363303010674 6566706666466567 |0063...temp...deg|
00000030  430100e70101000001 02               |C........|
```

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 0x33 to 0x34 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 0x33 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 0x34 is set to the integer temperature in tenths of degrees right shifted 7 bits. In this example 231 >> 7 = 0x01.

9. 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.

10. IANA Considerations

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

10.1. 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...
In addition to the units in this table, any of the Unified Code for Units of Measure [UCUM] in case sensitive form (c/s column) can be prepended by the string "UCUM:" and used in SEML.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>meter</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>kg</td>
<td>kilogram</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>s</td>
<td>second</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>A</td>
<td>ampere</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>K</td>
<td>kelvin</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>cd</td>
<td>candela</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>mol</td>
<td>mole</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>Hz</td>
<td>hertz</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>rad</td>
<td>radian</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>sr</td>
<td>steradian</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>N</td>
<td>newton</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>Pa</td>
<td>pascal</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>J</td>
<td>joule</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>W</td>
<td>watt</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>C</td>
<td>coulomb</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>V</td>
<td>volt</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>F</td>
<td>farad</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>Ohm</td>
<td>ohm</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>S</td>
<td>siemens</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>Wb</td>
<td>weber</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>T</td>
<td>tesla</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>H</td>
<td>henry</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>Cel</td>
<td>degrees Celsius</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>lm</td>
<td>lumen</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>lx</td>
<td>lux</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>Bq</td>
<td>becquerel</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>Gy</td>
<td>gray</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>Sv</td>
<td>sievert</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>kat</td>
<td>katal</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>pH</td>
<td>pH acidity</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>%</td>
<td>Value of a switch. A value of 0.0 indicates the switch is off while 100.0 indicates on.</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>count</td>
<td>counter value</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>%RH</td>
<td>Relative Humidity</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>m2</td>
<td>area</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>l</td>
<td>volume in liters</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>m/s</td>
<td>velocity</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>m/s2</td>
<td>acceleration</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>1/s</td>
<td>flow rate in liters per second</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>Units</td>
<td>Description</td>
<td>RFC-AAAA</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>W/m²</td>
<td>irradiance</td>
<td></td>
</tr>
<tr>
<td>cd/m²</td>
<td>luminance</td>
<td></td>
</tr>
<tr>
<td>Bspl</td>
<td>bel sound pressure level</td>
<td></td>
</tr>
<tr>
<td>bit/s</td>
<td>bits per second</td>
<td></td>
</tr>
<tr>
<td>lat</td>
<td>degrees latitude. Assumed to be in WGS84</td>
<td></td>
</tr>
<tr>
<td>lon</td>
<td>degrees longitude. Assumed to be in WGS84</td>
<td></td>
</tr>
<tr>
<td>%EL</td>
<td>remaining battery energy level in percents</td>
<td></td>
</tr>
<tr>
<td>EL</td>
<td>remaining battery energy level in seconds</td>
<td></td>
</tr>
<tr>
<td>beet/m</td>
<td>Heart rate in beets per minute</td>
<td></td>
</tr>
<tr>
<td>beats</td>
<td>Cumulative number of heart beats</td>
<td></td>
</tr>
</tbody>
</table>

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, Gd, sb, Lmb, ph, Cl, 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.

10.2. Media Type Registration

The following registrations are done following the procedure specified in [RFC4288] and [RFC3023].

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

10.2.1. 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 [RFC4627]. 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

10.2.2. 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

10.2.3. 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
10.3. 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. Security Considerations

See Section 12. Further discussion of security proprieties can be found in Section 10.2.

12. 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.

13. Acknowledgement

We would like to thank Lisa Dusseault, Joe Hildebrand, Lyndsay Campbell, Martin Thomson, John Klensin, Bjoern Hoehrmann, and Carsten Bormann for their review comments.

14. References
14.1. Normative References


[UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics.


14.2. Informative References


Authors’ Addresses

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA  95134
USA
Phone: +1 408 421-9990
Email: fluffy@cisco.com

Zach Shelby
ARM
150 Rose Orchard
San Jose  95134
FINLAND
Phone: +1-408-203-9434
Email: zach.shelby@arm.com
Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)  
draft-koster-core-coap-pubsub-01

Abstract

The Constrained Application Protocol (CoAP), and related extensions  
are intended to support machine-to-machine communication in systems  
where one or more nodes are resource constrained, in particular for  
low power wireless sensor networks. This document defines a publish-  
subscribe broker for CoAP that extends the capabilities of CoAP for  
supporting nodes with long breaks in connectivity and/or up-time.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal  
Provisions Relating to IETF Documents  
(http://trustee.ietf.org/license-info) in effect on the date of  
publishation of this document. Please review these documents  
carefully, as they describe your rights and restrictions with respect  
to this document. Code Components extracted from this document must
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-PubSub function set: A group of well-known REST resources that together provide the CoAP-PubSub service.

CoAP-PubSub Broker: A server node capable of receiving messages (publications) from and sending messages to other nodes, and able to match subscriptions and publications in order to route messages to the right destinations. The broker can also temporarily store publications to satisfy future subscriptions.

CoAP-PubSub Client: A CoAP client that implements the CoAP-PubSub function set.

Topic: A unique identifier for a particular item being published and/or subscribed to. A broker uses the topics to match subscriptions to publications.
3. Architecture

3.1. CoAP-PubSub Architecture

Figure 1 shows the architecture of a CoAP PubSub service. CoAP PubSub Clients interact with a CoAP PubSub Broker through the CoAP PubSub interface which is hosted by the Broker. State information is updated between the Clients and the Broker. The CoAP PubSub Broker performs a store-and-forward function of state updates between certain CoAP PubSub Clients. Clients Subscribe to state updates which are Published by other Clients, and which are forwarded by the Broker to the subscribing clients. The CoAP PubSub Broker also acts as a REST proxy, retaining the last state update provided by clients to supply in response to Read requests from Clients.

3.2. CoAP PubSub Broker

A CoAP PubSub Broker is a CoAP Server that exposes an interface for clients to use to initiate publish-subscribe interactions. Unlike clients, the broker needs to be reachable by all clients. The broker also needs to have sufficient resources (storage, bandwidth, etc.) to host CoAP resources, and potentially buffer messages, on behalf of the clients.

3.3. CoAP PubSub Client

A CoAP PubSub Client interacts with a CoAP PubSub Broker using the CoAP PubSub interface. Clients initiate all interactions with the CoAP-PubSub broker. A data source (e.g., sensor clients) can publish
state updates to the broker and data sinks (e.g., actuator clients) can read from or subscribe to state updates from the broker. Application clients can make use of both publish and subscribe in order to exchange state updates with data sources and sinks.

3.4. CoAP PubSub Topic

The clients and broker use topics to identify a particular resource or object in a publish-subscribe system. Topics are conventionally formed as a hierarchy, e.g. "/sensors/weather/barometer/pressure" or "EP-33543/sen/3303/0/5700". The topics are hosted at the broker and all the clients using the broker share the same namespace for topics.

4. CoAP PubSub Function Set

This section defines the interfaces between a CoAP PubSub Broker and PubSub Clients, which is called the CoAP PubSub Function Set. The examples throughout this section assume the use of CoAP [RFC7252]. A CoAP PubSub Broker implementing this specification MUST support the DISCOVER, CREATE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, READ, and REMOVE operations defined in this section. With the exception of PUBLISH, all operations in the CoAP PubSub Function Set MUST use confirmable (CON) CoAP messages.

4.1. DISCOVER

CoAP PubSub Clients discover CoAP PubSub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [I-D.ietf-core-resource-directory]. A CoAP PubSub Broker SHOULD indicate its presence and availability on a network by exposing a link to its PubSub function set at its .well-known/core location. A CoAP PubSub broker MAY register its PubSub function set location with a Resource Directory. Figure 2 shows an example of a client discovering a local PubSub Function Set using CoAP Simple Discovery. A broker wishing to advertise the CoAP PubSub Function Set for Simple Discovery or through a Resource Directory MUST use the link relation rt="core.ps".

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 PubSub 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.

Client                                          Broker
|                                               |
| ------ GET /.well-known/core?rt=core.ps ------->|
|                                               |
| <------2.05 Content "</ps>;rt=core.ps"--------|

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 PubSub function set location with a payload indicating the desired topic. The topic MUST use the CoRE link format [RFC6690]. 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. 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 := PubSub Function Set path (mandatory). The path of the PubSub Function Set, as obtained from discovery. A PubSub broker SHOULD use the value "ps" for this variable whenever possible.

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.

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

```
Client                                            Broker
|                                               |
| -------------- POST /ps "<topic1>" ----------> |
|                                               |
| <---------------- 2.01 Created ---------------|  
```

Figure 3: Example of CREATE

4.3. PUBLISH

A CoAP PubSub Client uses the PUBLISH interface for updating resources. The client MUST use the PUT method to publish state updates to the CoAP PubSub Broker. 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.
The Broker MUST publish updates to all clients subscribed on a particular topic each time it receives a publish on that topic. If a client publishes to a broker using non-confirmable messages, the broker MAY publish those messages to subscribed clients using non-confirmable messages. If a client publishes to a broker using confirmable messages, the broker MUST also publish those messages to subscribed clients using confirmable messages. If a client publishes to a broker with the max-age option, the broker MUST publish to subscribed clients including the same value for the max-age option. If a client publishes to a particular topic on a broker with a payload encoded in a particular CoAP Content-Format, the broker MUST publish to subscribed clients on that topic using the same CoAP Content-Format. A broker MUST use CoAP Notification as described in [I-D.ietf-core-observe] to publish to subscribed clients.

The PUBLISH interface is specified as follows:

Interaction: Client -> Broker

Method: PUT

URI Template: /{+ps}/{topic}

URI Template Variables:

  ps := PubSub Function Set path (mandatory). The path of the PubSub Function Set, as obtained from discovery.
  topic := The desired topic to publish on.

Content-Format: Any valid CoAP Content Format

Payload: Representation of the topic value (CoAP resource state representation) in the indicated Content-Format

The following response codes are defined for this interface:

Success: 2.04 "Changed". Successful publish, topic is updated

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

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.
4.4. SUBSCRIBE

CoAP PubSub Clients subscribe to topics on the Broker using CoAP Observe as described in [I-D.ietf-core-observe]. A CoAP PubSub Client wishing to Subscribe to a topic on a broker MUST use a CoAP GET with Observe registration. The Broker MAY add the client to a list of observers. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value. 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 SUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

URI Template: /{+ps}/{topic}

URI Template Variables:

ps := PubSub Function Set path (mandatory). The path of the PubSub Function Set, as obtained from discovery.

topic := The desired topic to subscribe to.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful subscribe, current value included
Success: 2.04 "No Content". Successful subscribe, value not included.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 5 shows an example of Client2 subscribing to "topic1" and receiving two publish responses from the broker. The first publish from the broker uses the last stored value associated with the topic1. The second publish from the broker is in response to the publish received from Client1.

<table>
<thead>
<tr>
<th>Client1</th>
<th>Client2</th>
<th>Broker</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>----- GET /ps/topic1 Observe:0 Token:XX ----&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;-------- 2.05 Content Observe:10----------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;-------- 2.05 Content Observe:11----------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;-------- PUT /ps/topic1 &quot;1033.3&quot; --------&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;-------- 2.05 Content Observe:11----------</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

CoAP PubSub Clients unsubscribe from topics on the Broker using the CoAP Cancel Observation operation. A CoAP PubSub Client wishing to unsubscribe to a topic on a Broker MUST either use CoAP GET with Observe using an Observe parameter of 1 or send a CoAP Reset message in response to a publish [I-D.ietf-core-observe].

The UNSUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1
URI Template:  /{+ps}/{topic}

URI Template Variables:

ps := PubSub Function Set path (mandatory). The path of the 
PubSub Function Set, as obtained from discovery.

topic := The desired topic to unsubscribe from.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful unsubscribe, current value included

Success: 2.04 "No Content". Successful unsubscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 6 shows an example of a client unsubscribe using the Observe=1 cancellation method.

Client                                          Broker

| ----- GET /ps/topic1 Observe:1 Token:XX ----> |
<---------------------- 2.05 Content ----------------------|

Figure 6: Example of UNSUBSCRIBE

4.6. READ

A CoAP PubSub client wishing to obtain only the most recent published value on a topic MAY use the READ interface. For reading, the client uses the CoAP GET method. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value. 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 READ interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: /{+ps}/{topic}

URI Template Variables:

ps := PubSub Function Set path (mandatory). The path of the
PubSub Function Set, as obtained from discovery.

topic := The desired topic to READ.

The following response codes are defined for this interface:

Success: 2.05 "Content". Successful READ, current value included
Success: 2.04 "No Content". Topic exists, value not included
Failure: 4.00 "Bad Request". Malformed request.
Failure: 4.01 "Unauthorized". Authorization failure.
Failure: 4.04 "Not Found". Topic does not exist.

Figure 7 shows an example of a successful READ from topic1.

Client                                Broker
--------------------------------------
<----------------- GET /ps/topic1 ------>|
|                                    |
|                                    |
| <----------------- 2.05 Content ------|

Figure 7: Example of READ
4.7. REMOVE

A CoAP PubSub Client wishing to remove a topic can use the CoAP Delete operation on the URI of the topic. The CoAP PubSub Broker MUST return "2.02 Deleted" if the remove operation is successful. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if the topic can not be removed.

The REMOVE interface is specified as follows:

Interaction:  Client -> Broker

Method:  DELETE

URI Template:  /{+ps}/{topic}

URI Template Variables:

ps := PubSub Function Set path (mandatory). The path of the PubSub Function Set, as obtained from discovery.

topic := The desired topic to REMOVE.

Content-Format:  None

Response Payload:  None

The following response codes are defined for this interface:

Success:  2.02 "Deleted". Successful remove

Failure:  4.00 "Bad Request". Malformed request.

Failure:  4.01 "Unauthorized". Authorization failure.

Failure:  4.04 "Not Found". Topic does not exist.

Figure 5 shows a successful remove of topic1.

A CoAP PubSub Client may register CoRE Links [RFC6690] to create PubSub Topics with an RD. A PubSub Client may use an RD to discover PubSub Topics. A client which registers PubSub Topics with an RD MUST use the context relation (con) [I-D.ietf-core-resource-directory] to indicate that the context of the registered links is the PubSub Broker.

6. Sleep-Wake Operation

CoAP PubSub provides a way for client nodes to sleep between operations, conserving energy during idle periods. This is made possible by shifting the server role to the broker, allowing the broker to be always-on and respond to requests from other clients while a particular client is sleeping.

For example, the broker will retain the last state update received from a sleeping client, in order to supply the most recent state update to other clients in response to read and subscribe operations.

Likewise, the broker will retain the last state update received on the topic such that a sleeping client, upon waking, can perform a read operation to the broker to update its own state from the most recent system state update.

7. Security Considerations

CoAP-PubSub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP-PubSub broker and the clients SHOULD authenticate each other and enforce access control.
policies. A malicious client could subscribe to data it is not authorized to or mount a denial of service attack against the broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP-PubSub broker introduces challenges for the use of end-to-end security between for example a client device on a sensor network and a client application running in a cloud-based server infrastructure since brokers terminate the exchange. While running separate DTLS sessions from the client device to the broker and from broker to client application protects 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-PubSub broker, the use of end-to-end object security is RECOMMENDED [I-D.selander-ace-object-security].

8. 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.

8.1. Resource Type value 'core.ps'

- Attribute Value: core.ps
- Description: Section 4 of [[This document]]
- Reference: [[This document]]
8.2. 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. 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.

10. References

10.1. Normative References

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

[I-D.ietf-core-resource-directory]

[I-D.selander-ace-object-security]


Informative References


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
Abstract

JSON (RFC7159) is a text-based data format which is popular for Web based data exchanges. CBOR (RFC7049) is a binary data format which has been optimized for data exchanges for the Internet of Things. For many IoT scenarios, CBOR formats will be preferred since it can decrease transmission payload sizes compared to other data formats.

This specification defines an approach for translating JSON objects, which are relevant for the CoRE WG and its related specifications, into CBOR format. Where applicable, mapping from other formats into JSON or CBOR is also described.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

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

1.1. Objectives

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 particularly 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 its efficiency.

When converting between JSON and CBOR, as usual, 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 JSON objects into CBOR format which are relevant for the CoRE WG and its related specifications. Where applicable, mapping from other formats (e.g. CoRE Link Format) into JSON or CBOR is also described.

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].

CoAP: Constrained Application Protocol
CBOR: Concise Binary Object Representation
CoRE: Constrained RESTful Environments
IoT: Internet of Things
JSON: JavaScript Object Notation

2. Web Links in 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]. [I-D.ietf-core-links-json] defines a common format for representing Web links in JSON format.

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".
The URI-Reference is represented as a name/value pair with the name "href" and the URI-Reference as the value.

The link attributes are also represented as name/value pairs with attribute names and attribute values.

The information model of the CoRE Link Format can be summarized below:

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>href</td>
<td>resource URI</td>
</tr>
<tr>
<td>attribute name 1</td>
<td>attribute value 1</td>
</tr>
<tr>
<td>attribute name 2</td>
<td>attribute value 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>attribute name n</td>
<td>attribute value n</td>
</tr>
</tbody>
</table>

Figure 1: CoRE Link Format Information Model

2.3. Encoding Step

To reduce message size, it would be beneficial to perform an extra encoding step, and encode "href" and the standardized attribute names specified in [RFC5988] and [RFC6690] as integers.

The encoding is summarized below:
<table>
<thead>
<tr>
<th>name</th>
<th>encoded value</th>
</tr>
</thead>
<tbody>
<tr>
<td>href</td>
<td>1</td>
</tr>
<tr>
<td>rel</td>
<td>2</td>
</tr>
<tr>
<td>anchor</td>
<td>3</td>
</tr>
<tr>
<td>rev</td>
<td>4</td>
</tr>
<tr>
<td>hreflang</td>
<td>5</td>
</tr>
<tr>
<td>media</td>
<td>6</td>
</tr>
<tr>
<td>title</td>
<td>7</td>
</tr>
<tr>
<td>type</td>
<td>8</td>
</tr>
<tr>
<td>rt</td>
<td>9</td>
</tr>
<tr>
<td>if</td>
<td>10</td>
</tr>
<tr>
<td>sz</td>
<td>11</td>
</tr>
<tr>
<td>ct</td>
<td>12</td>
</tr>
</tbody>
</table>

Figure 2: Link Attributes Encoding

2.4. Mapping

The objective of the mapping defined in this section is to map information from the JSON formats specified in [RFC6690] into CBOR format.

We straightforwardly map:

- the outer collection to an array of links (Major type 4)
- each link to a map of pairs of data items (Major type 5)

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a CBOR pair of data items.

The URI-Reference is represented as a name/value pair with the name "href" and the URI-Reference as the value.
2.4.1. Link Format to CBOR Example

This examples shows conversion from link format to CBOR format.

```
</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"
```

becomes

```
85                                     # array(number of data items:5)
a3                                  # map(number of pairs of data items: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(number of pairs of data items:3)
    01                               # unsigned integer(value:1, "href")
    6d                               # text string(13 bytes)
      2f73656e736f72732f74656670    # "/sensors/temp"
    09                               # unsigned integer(value:9,"rt")
    6d                               # text string(13 bytes)
      74656d706572617475726552d63  # "temperature-c"
    0a                               # unsigned integer(value:10,"if")
    66                               # text string(6 bytes)
      73656e736f72              # "sensor"
  a3                                  # map(number of pairs of data items:3)
    01                               # unsigned integer(value:1, "href")
    6e                               # text string(14 bytes)
      2f73656e736f72732f6c69676874  # "/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(number of pairs of data items:3)
    01                               # unsigned integer(value:1, "href")
```

Figure 3: Example from page 15 of [RFC6690]
2.4.2. Link Format in JSON to CBOR Example

This example shows conversion from link format JSON to CBOR format.

```
[{
  "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"
}]
```

Figure 5: Example from section 2.1 of [draft-ietf-core-links-json] becomes

85
a3
01
60
2f73656e736f72732f7465667073656e736f72732f74656670
0c
18 28
07
Figure 6: Web Links Encoded in CBOR

3. Group Communication Management Objects in CBOR

3.1. Background

The CoAP Group Communications (RFC7390) defines management objects in JSON format. These objects are used to represent IP multicast group information for CoAP endpoints.

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:

```
+------------------+--------------------+
| name             |   value            |
+------------------+--------------------+
| n                | group name         |
+------------------+--------------------+
| a                | IP multicast       |
|                  | address            |
+------------------+--------------------+
```

Figure 7: CoAP Group Communication Information 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.

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 8: 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)
    5b666631353a3a343230303a663766653a656433373a313463615d # "[ff15::4200:f7fe:ed37:14ca]"
  62                                   # text(2)
   3131                              # "11"
  a2                                   # map(2)
   61                                # text(1)
   6e                             # "n"
  78 25                             # text(37)
    73656e736f72732e666c6f6f72312e776573742e626c6467362e6578616d706c652e636f6d # "sensors.floor1.west.bldg6.example.com"
  61                                # text(1)
  61                                # "a"
  78 1b                             # text(27)
    5b666631353a3a343230303a663766653a656433373a323563625d # "[ff15::4200:f7fe:ed37:25cb]"
  62                                   # text(2)
   3132                              # "12"
  a2                                   # map(2)
   61                                # text(1)
   6e                             # "n"
  78 29                             # text(41)
    416c6c2d446576696365732e666c6f6f72312e776573742e626c6467362e6578616d706c652e636f6d # "All-Devices.floor1.west.bldg6.example.com"
  61                                # text(1)
  61                              # "a"
  78 20                             # text(32)
    5b666631353a3a343230303a663766653a656433373a616263645d3a34353637 # "[ff15::4200:f7fe:ed37:abcd]:4567"

Figure 9: Group Communication Management Object Encoded in CBOR
4. IANA Considerations

This specification registers the following additional Internet Media Types:

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 [RFC7159], Section 6.

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: Kepeng Li
<kepeng.lkp@alibaba-inc.com>

Intended usage: COMMON

Change controller: IESG

5. Security Considerations

The security considerations of [RFC6690] and [RFC7049] apply.

6. Acknowledgements

Special thanks to Bert Greevenbosch who was an author on the initial version of this document.
7. References

7.1. Normative References


7.2. Informative References

[I-D.ietf-core-links-json]
Bormann, C., "Representing CoRE Link Collections in JSON", draft-ietf-core-links-json-02 (work in progress), July 2014.

Authors’ Addresses

Kepeng LI
Alibaba Group
Wenyixi Road, Yuhang District
Hangzhou, Zhejiang 311121
China

Email: kepeng.lkp@alibaba-inc.com
CoAP over WebSockets
draft-savolainen-core-coap-websockets-03

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on April 10, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

```
coap+ws://example.org/sensors/temperature?u=Cel
```

Uri-Path: "sensors"

```
ws://example.org/.well-known/coap
```

Uri-Path: "temperature"

```
coap+ws://example.org/sensors/temperature?u=Cel
```

Uri-Query: "u=Cel"

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)

In a completely different way, another 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 [I-D.dijk-core-sleepy-reqs].

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

The challenge, again, is to identify the resource. Since the CoAP server is running inside the web browser, this requires not only to identify the WebSocket client and the path and query, but also the intermediary, which is the only path to reach the server. The
problem can be avoided if the intermediary is turned into a Reverse Proxy or a Mirror Server [I-D.vial-core-mirror-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: dGhlIHNhbXBsZXBsZSBub25jZQ==
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.

The resulting message format is shown in Figure 7. The four most-significant bits of the first byte are reserved (R). The remaining fields and structure are the same as defined in [RFC7252].

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   R   |  TKL  |      Code     |    Token (TKL bytes) ... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: CoAP Message Format over WebSockets

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, a multiplexing extension such as [I-D.ietf-hybi-websocket-multiplexing] could be used. Alternatively, 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 locally scoped 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 [I-D.ietf-core-observe] 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), 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 an "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. Acknowledgements

Thanks to Nadir Javed for helpful comments and discussions that have shaped the document.

7. References

7.1. Normative References

[I-D.ietf-core-observe]
   Hartke, K., "Observing Resources in CoAP",
   draft-ietf-core-observe-14 (work in progress), June 2014.

[RFC2119]  Bradner, S., "Key words for use in RFCs to Indicate

7.2. Informative References


[I-D.becker-core-coap-sms-gprs]

[I-D.dijk-core-sleepy-reqs]

[I-D.ietf-core-block]

[I-D.ietf-hybi-websocket-multiplexing]

[I-D.vial-core-mirror-server]


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
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
Object Security for CoAP

draft-selander-ace-object-security-01

Abstract

This memo presents a scheme for data object security applicable to protection of payload of generic message formats as well as request and response messages of the Constrained Application Protocol (CoAP).

Status of this Memo

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

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

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

Copyright and License Notice

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

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

Table of Contents

1. Introduction ................................................. 4
  1.1 Terminology ............................................. 4
2. Background ................................................... 5
3. End-to-end Security in Presence of Intermediary Nodes .............. 6
4. Secure Message ............................................... 7
  4.1 Secure Message format ................................... 8
    4.1.1 Secure Message Header .............................. 8
    4.1.2 Secure Message Body ................................. 9
      4.1.2.1 Secure Signed Message Body ....................... 9
      4.1.2.2 Secure Encrypted Message Body ................. 9
    4.1.3 Secure Message Tag .................................. 9
4. Secure Message ............................................... 7
5. Message Protection .......................................... 9
  5.1 CoAP Message Protection .................................. 10
    5.1.1 The Sig Option ..................................... 10
      5.1.1.1 Option Structure ................................ 10
      5.1.1.2 Integrity Protection and Verification ........ 11
      5.1.1.3 SSM Body ....................................... 11
    5.1.2 The Enc Option ..................................... 12
      5.1.2.1 Option Structure ................................ 12
      5.1.2.2 Encryption and Decryption ..................... 12
      5.1.2.3 SEM Body ...................................... 13
      5.1.2.4 CoAP Message with Enc Option ................. 13
    5.1.3 SM Tag ............................................... 14
5. Application Layer Protection ................................ 14
5.3 Replay Protection and Freshness ................................ 15
  5.3.1 Replay Protection .................................... 15
  5.3.2 Freshness ............................................ 15
6. Security Considerations ....................................... 16
7. Privacy Considerations ....................................... 17
8. IANA Considerations ......................................... 18
9. Acknowledgements ........................................... 18
10. References .................................................. 19
  10.1 Normative References .................................... 19
  10.2 Informative References .................................. 19
Appendix A. Which CoAP Header Fields and Options to Protect ........... 20
  A.1 CoAP Header Fields ...................................... 20
  A.2 CoAP Options ............................................ 20
    A.2.1 Integrity Protection ............................... 21
A.2.2 Encryption ............................................. 22
A.2.3 Summary ................................................. 22
Appendix B. JOSE Objects as Secure Messages ................. 23
  B.1 JWS as Secure Signed Message ...................... 23
  B.2 JWE as Secure Encrypted Message ................. 23
  B.3 "seq" (Sequence Number) Header Parameter ............ 24
  B.4 "mod" (Mode) Header Parameter .................. 24
Appendix C. Compact Secure Message ...................... 25
  C.1 CSM Format ........................................... 25
  C.2 Comparison of Secure Message sizes ................ 27
Appendix D. Examples ......................................... 29
  D.1 CoAP Message Protection .......................... 29
    D.1.1 Integrity protection of CoAP Message ........ 29
    D.1.2 Encryption of CoAP Message .................. 30
  D.2 Application Layer Protection ..................... 32
    D.2.1 Proxy Caching ................................ 32
  D.2.2 Publish-Subscribe ............................... 33
    D.2.3 Transporting Authorization Information ....... 34
Authors' Addresses ........................................... 35
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 commonly used features of CoAP are store-and-forward and publish-subscribe exchanges, which 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. On the other hand, 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 presents an object security approach for secure messaging in constrained environments that may be used as a complement to DTLS for store-and-forward and publish-subscribe CoAP exchanges. Note that the solution sketched in this memo can 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 messages during transport between end-point and intermediary node.

This version of the draft focuses on symmetric key based algorithms. Public key based algorithms will be addressed in the next version.

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 RFC 4949 [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", "signature", and "verify".

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].

Client, Resource Server, and Authorization Server are defined in [I-D.seitz-ace-problem-description]. The terms "server" and "Resource Server" are used interchangeably.

JSON Web Signature (JWS), JOSE Header, JWS Payload, and JWS Signature are defined in [I-D.ietf-jose-json-web-signature].

JSON Web Encryption (JWE), JWE AAD, JWE Ciphertext, and JWE Authentication Tag are defined in [I-D.ietf-jose-json-web-encryption].

Secure Message (SM), Secure Signed Message (SSM), and Secure Encrypted Message (SEM) are message formats defined in this memo. The Compact Secure Message (CSM) format is defined in Appendix C. The Sig and Enc options are CoAP options defined in this memo.

Excluded Authenticated Data (EAD) is defined in this memo (see Sections 4.1.2). Transaction Identifier (TID) is defined in this memo (see Section 4.1.1).

2. Background

The background for this work is provided by the use cases and problem description in [I-D.ietf-ace-usecases] and [I-D.seitz-ace-problem-description]. The overall objective is that (a) only authorized requests are granted, and (b) messages between client and server are protected (according to requirements of the particular use case). The focus of this memo is on end-to-end security in constrained environments in the presence of intermediary nodes, which corresponds to point (b).

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.

On a high level, end-to-end security in this setting encompasses:

1. Protection against eavesdropping and manipulation of resource
representations in intermediary nodes;

2. Protection against message replay;

3. Protection of authorization information ("access tokens") in transport from an Authorization Server to a Resource Server via a Client, or other intermediary nodes which could gain from changing the information;

4. Allowing a client to verify that a response comes from a certain server and is the response to a particular request;

5. Protection of the RESTful method used by the client, or the response code used by the server. For example if a malicious proxy replaces the client requested GET with a DELETE this must be detected by the server;

6. Protection against eavesdropping of meta-data of the request or response, including CoAP options such as for example Uri-Path and Uri-Query, which may reveal some information on what is requested.

From the listed examples, there are two main categories of security requirements and corresponding solutions. The first category deals essentially with application layer protection, i.e. protecting the payload of the RESTful protocol (1-3). The second category deals with protecting an entire CoAP message, targeting also CoAP options and header fields (4-6). The next section formulates security requirements for the two categories, which are denoted Mode:APPL and Mode:COAP, respectively.

3. End-to-end Security in Presence of Intermediary Nodes

For high-level security requirements related to resource access, see section 4.6 of [I-D.seitz-ace-problem-description]. This section defines the specific requirements that address the two categories of examples identified in the previous section, taking into account potential intermediary nodes.

In the case of application layer protection (Mode:APPL), the end-to-end security requirements apply to the RESTful protocol payload data, such as Resource Representations:

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

b. It shall be possible for an intended receiver to detect if it has received this message previously, i.e. replay protection.
In this case there may be multiple receivers of a given message, for example in the case of a proxy that is caching responses used to serve multiple clients, or in a publish-subscribe setting with multiple subscribers to a given publication.

In the case of protecting specific Client-Server CoAP message exchanges (Mode: COAP), potentially passing via intermediary nodes, there are additional end-to-end security requirements:

- c. The CoAP options which are not intended to be changed by an intermediary node shall be integrity protected between Client and Server.
- d. The CoAP options which are not intended to be read by an intermediary node shall be encrypted between Client and Server.
- e. The CoAP header field "Code" shall be integrity protected between Client and Server.
- f. A Client shall be able to verify that a message is the response to a particular request the Client made.

The requirements listed above can be met by encryption, integrity protection and replay protection. What differs is the actual data that is protected, i.e. application layer data or CoAP message data. This memo specifies a common "Secure Message" format that can be used to wrap either payload only or also additional selected CoAP message fields, and be sent as part of the message.

4. Secure Message

There exist already standardized and draft content formats for cryptographically protected data such as CMS [RFC5652], JWS, JWE, and COSE [I-D.bormann-jose-cose]. None of the listed formats provide support for replay protection, but it is noted in section 10.10 of [I-D.ietf-jose-json-web-signature]) that one way to thwart replay attacks is to include a unique transaction identifier and have the recipient verify that the message has not been previously received or acted upon.

The term Secure Message (SM) format refers to a content format for cryptographically protected data which includes a unique transaction identifier and allows customization to support different variants of format and message processing (Modes).

This memo uses JOSE content formats as a model to specify format and processing of messages. The terms Secure Signed Message (SSM) format
and Secure Encrypted Message (SEM) format to refer to Secure Message formats supporting integrity protection only and additional encryption, analogous to JWS and JWE, respectively. Appendix B shows how JWS and JWE could be extended to become Secure Message formats.

It should be noted that the current JOSE objects are undesirably large for very constrained devices. In their current size they can lead to packet fragmentation in constrained-node networks due to limited frame sizes, and to problems with limited storage capacity on constrained devices due to limited RAM. COSE renders more compact objects, and further optimizations are considered. See Appendix C for a discussion of minimum message expansion and message format overhead.

4.1 Secure Message format

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

4.1.1 Secure Message Header

The following parameters SHALL be included in the SM Header:

- **Algorithm.** This parameter allows the receiver to identify the cryptographic algorithm(s) used to protect the Secure Message. In case of SSM it has the same syntax as the JOSE Header Parameter "alg" defined in Section 4.1.1 of [I-D.ietf-jose-json-web-signature]. In case of SEM, it has the same syntax as the JOSE Header Parameter "enc" defined in Section 4.1.2 of [I-D.ietf-jose-json-web-encryption]. (Assuming direct key agreement, corresponding to the JWE "alg" = "dir" setting.)

- **Key Identifier.** This parameter allows the receiver to uniquely identify the sender and the security context/key(s) used with the Algorithm. It has the same syntax as the JOSE Header Parameter "kid" defined in Section 4.1.4 of [I-D.ietf-jose-json-web-signature].

- **Sequence Number.** The Sequence Number parameter enumerates the Secure Messages protected using the key(s) identified by the Key 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.

- **Mode.** The Mode parameter defines application specific message format, content and processing. This parameter provides means for customization of the Secure Message format, in particular to distinguish between Secure Messages containing application layer data only or CoAP message data.
The ordered sequence (Sequence Number, Key Identifier) is called Transaction Identifier (TID), and SHALL be unique for each SM.

4.1.2 Secure Message Body

Analogously to JWS and JWE, the SM Body contains what is being protected. The SM Body is different for SSM and SEM.

In order to obtain a compact representation, certain data is integrity protected but excluded from the Secure Message. Such data is referred to as Excluded Authenticated Data (EAD). To further reduce message size, the unencrypted part of the SM Body may be "detached" from the Secure Message, see sections 4.1.2.1 and 4.1.2.2.

The assumption behind excluding integrity protected data from the SM, or detaching integrity protected but not encrypted parts of the SM during transport, is that the data in question is known to the receiver, e.g. because it is exchanged beforehand or because it is transported as part of the CoAP message carrying the Secure Message.

4.1.2.1 Secure Signed Message Body

For SSM, the Body consists of the payload data which is integrity protected, analogously to the JWS Payload. Detached Content is defined to mean that the Body is removed from the Secure Message, analogously to Appendix F of [I-D.ietf-jose-json-web-signature]. Hence a SSM with Detached Content consists of Header and Tag.

4.1.2.2 Secure Encrypted Message Body

Analogously to JWE, the terms Plaintext, Ciphertext and Additional Authenticated Data (AAD) are used for the SEM. The Body of a SEM consists of Ciphertext and Additional Authenticated Data (AAD). For SEM Detached Content is defined to mean that the AAD is removed from the Secure Message. Hence a SEM with Detached Content consists of the Header, Ciphertext and Tag.

4.1.3 Secure Message Tag

The SM Tag consists of the Signature / Authentication Tag value as defined by the Algorithm, calculated over the SM Header, SM Body and EAD (if present). The content of EAD depends on the Mode, see 5.1.3 and 5.2

5. Message Protection

This section describes what is protected in a Secure Message and how it depends on the defined Modes ("CoAP Message Protection" and "Application Layer Protection"). Both formats SSM and SEM defined in the previous section are applicable to both Modes. For examples, see Appendix D.

For any Secure Message Mode, the SEM format SHALL be used by default. The SM Header is defined in 4.1.1, indicates the Mode, but is in all other respects handled similarly in both Modes. This section also describes the differences in SM Body and SM Tag.

5.1 CoAP Message Protection

Referring to examples 4-6 in Section 2 and requirements a-f in Section 3, this section presents how to protect individual CoAP messages including options and header fields, as well as request-response message exchanges, using the Secure Message format. This is called Secure Message Mode:COAP. An endpoint receiving a CoAP request containing a Secure Message with Mode:COAP MUST respond with a CoAP message containing a Secure Message with Mode:COAP.

Since slightly different message formats are used for integrity protection only (SSM), and additional encryption (SEM), these cases are treated separately. Two new CoAP security options are introduced: the Enc option and the Sig option. A CoAP message SHALL NOT include both Enc and Sig options.

5.1.1 The Sig Option

In order to integrity protect CoAP message exchanges, a new CoAP option is introduced: the Sig option, containing a SSM Mode:COAP object. Endpoints supporting this scheme MUST check for the presence of a Sig option, and verify the SSM as described in Section 5.1.1.2 before accepting a message as valid.

5.1.1.1 Option Structure

The Sig option indicates that certain CoAP Header Fields, Options, and Payload (if present) are integrity and replay protected using a Secure Signed Message (SSM). The Sig option SHALL contain a SSM with Detached Content (see Section 4.1.2.1).

This option is critical, safe to forward, it is not part of a cache key, and it is not repeatable. Table 1 illustrates the structure of this option.
### 5.1.1.2 Integrity Protection and Verification

A CoAP endpoint composing a message with the Sig option SHALL process the SSM and produce the SSM Tag, as defined in 5.1.1.3 and 5.1.3, analogously to the specification for producing a JWS object as described in Section 5.1 of [I-D.ietf-jose-json-web-signature] (cf. Appendix B). In addition, the sending endpoint SHALL process the Sequence Number as described in Section 5.3.

A CoAP endpoint receiving a message containing the Sig option SHALL first recreate the SSM Body as described in Section 5.1.1.3, and then verify the SSM Tag as described in Section 5.1.3, analogously to the specification for verifying a JWS object as described in Section 5.2 of [I-D.ietf-jose-json-web-signature] (cf. Appendix B). In addition, the receiving endpoint SHALL process the Sequence Number as described in Section 5.3.

NOTE: The explicit steps of the protection and verification procedure will be included in a future version of this draft.

### 5.1.1.3 SSM Body

The SSM Body SHALL consist of the following data, in this order:

- the 8-bit CoAP header field Code;
- all CoAP options present which are marked as IP in Table 3 (Appendix A), 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 CoAP Payload (if any).
5.1.2 The Enc Option

In order to encrypt and integrity protect CoAP messages, a new CoAP option is introduced: the Enc option, indicating the presence of a SEM Mode:COAP object in the CoAP message, containing the encrypted part of the CoAP message. Endpoints supporting this scheme MUST check for the presence of an Enc option, and verify the SEM as described in 5.1.2.2 before accepting a message as valid.

NOTE: This version of the draft is only considering AEAD algorithms.

5.1.2.1 Option Structure

The Enc option indicates that certain CoAP Options and Payload (if present) are encrypted, integrity and replay protected using a Secure Encrypted Message (SEM) with Detached Content (see Section 4.1.2.2). The structure of a CoAP message with an Enc option is described in Section 5.1.2.4.

This option is critical, safe to forward, it is not part of a cache key, and it is not repeatable. Table 2 illustrates the structure of this option.

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>Enc</td>
<td>opaque</td>
<td>0 or 12-TBD</td>
</tr>
</tbody>
</table>

Table 2: The Enc Option

*) Length indicates in this case the additional length added to the total length of all CoAP options. If the CoAP message has Payload, then the Enc option is empty, otherwise it contains the SEM (see Section 5.1.2.4). In the latter case, the SEM Ciphertext contains the encrypted CoAP Options (see Section 5.1.2.3), which are thus excluded from plaintext part of the message. Hence the additional length is essentially Length(SEM Header) + Length(SEM Tag). The minimum length is estimated in Appendix C. The maximum length depends on actual message format selected and is TBD.

5.1.2.2 Encryption and Decryption

A CoAP endpoint composing a message with the Enc option SHALL process
the SEM and produce the SEM Ciphertext and SEM Tag, as defined in
5.1.2.3 and 5.1.3, analogously to the specification for producing a
JWE object as described in Section 5.1 of [I-D.ietf-jose-json-web-
encryption] (cf. Appendix B). In addition, the sending endpoint
SHALL process the Sequence Number as described in Section 5.3.

A CoAP endpoint receiving a message containing the Enc option SHALL
first recreate the SEM Body as described in Section 5.1.2.3, and then
decrypt and verify the SEM analogously to the specification for
verifying a JWE object as described in Section 5.2 of [I-D.ietf-jose-
json-web-encryption] (cf. Appendix B). In addition, the receiving
endpoint SHALL process the Sequence Number as described in Section
5.3.

NOTE: The explicit steps of the protection and verification procedure
will be included in a future version of this draft.

5.1.2.3 SEM Body

The SEM 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 Table 3 (see
  Appendix A), 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).

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

- the 8-bit CoAP header field Code;

- all CoAP options present which are marked as IP and not marked
  as E in Table 2 (see Appendix A), in the order as given by the
  Option number (each Option with Option Header including delta to
  previous such Option).

5.1.2.4 CoAP Message with Enc Option

An unprotected CoAP message is encrypted and integrity protected by
means of an Enc option and a SEM. The structure and format of the
protected CoAP message being sent instead of the unprotected CoAP
message is now described.
The protected CoAP message is formatted as an ordinary CoAP message, with the following Header, Options and Payload:

- 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, and the Enc 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 Enc option SHALL contain the SEM with Detached Content. If the unprotected CoAP message has Payload, then the SEM option SHALL be empty and the Payload of the CoAP message SHALL be the SEM with Detached Content. The Payload is prefixed by the one-byte Payload Marker (0xFF).

5.1.3 SM Tag

This section describes the SM Tag for Mode:COAP, which applies both to SEM and SSM. The SM Tag is defined in 4.1.3. If the message is a CoAP Request, then EAD SHALL be empty. If the message is a CoAP Response, then EAD SHALL consist of the TID of the associated CoAP Request.

5.2 Application Layer Protection

Referring to examples 1-3 in Section 2 and requirements a and b in Section 3, the case of only protecting Payload sent in a RESTful protocol using the Secure Message format is now discussed. This is called Secure Message Mode:APPL.

The sending endpoint SHALL wrap the Payload, and the receiving endpoint unwrap the Payload in the relevant SM format (SSM or SEM) Mode:APPL. The SSM (SEM) SHALL be protected (encrypted) and verified (decrypted) as described in 5.1.1.2 (5.1.2.2), including replay protection as described in section 5.3.

NOTE: The explicit steps of the protection and verification procedure will be included in a future version of this draft.

For Mode:APPL, the EAD SHALL be empty. Hence, the SM Tag is calculated over the SM Header and SM Body.

A CoAP message where the Payload is wrapped as a Secure Message...
5.3 Replay Protection and Freshness

In order to protect from replay of messages and verify freshness of responses, a CoAP endpoint SHALL maintain Transaction Identifiers (TIDs) of sent and received Secure Messages (see section 4.1.1).

5.3.1 Replay Protection

An endpoint supporting Secure Message SHALL maintain two TIDs and associated security context/key(s) for each other endpoint it communicates with, one TID for protecting sent messages, and one TID for verifying the received messages. Depending on use case, an endpoint MAY maintain a sliding receive window for Sequence Numbers associated to TIDs in received messages, equivalent to the functionality described in section 4.1.2.6 of [RFC6347].

Before composing a new message a sending endpoint supporting Secure Message SHALL step the Sequence Number of the associated send TID and SHALL include it in the SM Header parameter Sequence Number as defined in section 4.1.1. However, if the Sequence Number counter wraps, the client must first acquire a new TID and associated security context/key(s). The latter is out of scope of this memo.

A receiving endpoint supporting Secure Message SHALL verify that the Sequence Number received in the SM Header is greater than the Sequence Number in the TID for received messages (or within the sliding window and not previously received) and update the TID (window) accordingly.

5.3.2 Freshness

If a CoAP server receives a valid Secure Message request in Mode:COAP, then the response SHALL include the TID of the request as EAD, as defined in section 5.1.3. If the CoAP client receives a Secure Message response in Mode:COAP, then the client SHALL verify the signature by reconstructing SM Body and using the TID of its own associated request as EAD, as defined in section 5.1.3.
6. Security Considerations

In scenarios with proxies, gateways, or caching, DTLS only protects data hop-by-hop meaning that all 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 this proposal only protects selected message fields. DTLS, however, also incurs a large overhead cost, e.g. due to the handshake procedure. While that cost can be amortized in scenarios with long lived connections, in cases where a device will have connections with varying clients, using secured objects instead of session security can provide a significant performance gain.

Secure Message Mode: COAP addresses point to point encryption, integrity and replay protection, and freshness of response. Payload as well as relevant options and header field Code are protected. It is possible to define unique session keys to enable perfect forward secrecy.

Secure Message Mode: APPL only protects payload and only gives replay protection (not freshness), but this allows more use cases such as point to multi-point 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.

Using blockwise transfer [I-D.ietf-core-coap-block], the integrity protection as provided by the method described here only covers the individual blocks, not the entire request or response. One way to handle this would to allow the Sig or Enc option to be repeatable, and in one or several of the block transfer carry a MAC or signature that covers the entire request or response.

The Version header field is not integrity protected to allow backwards compatibility with future versions of CoAP. Considering this, it may in theory be possible to launch a
cross-version attack, e.g. something analogously to a bidding down attack. Future updates of CoAP would need to take this into account.

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.

Independently of message format, and whether the target is application layer protection or CoAP message protection, 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.seitz-ace-problem-description]. 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 used in the Secure Message exchange.

The security contexts required for SSM and SEM are different. For a SSM, the security context is essentially Algorithm, Key Identifier, Sequence Number and Key. For a SEM it is also required to have a unique AEAD Initialization Vector for each message. The AEAD Initialization Vector SHALL be the concatenation of a Salt (8 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. For example, the Salt may be established by the same means as the keys used to secure the protocol between the sender and receiver. For a SEM, the security context is essentially Algorithm, Key Identifier, Salt, Sequence Number and Key.

NOTE: This last paragraph will be moved into the main document in a future version of this draft.

7. 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 options provides
additional protection as to the content and nature of the message exchange.

The headers sent in plaintext allows 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).

8. 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:

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>Sig</td>
<td>[this document]</td>
</tr>
<tr>
<td>TBD</td>
<td>Enc</td>
<td>[this document]</td>
</tr>
</tbody>
</table>

NOTE: IANA considerations for Mode is TBD

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

Media Type: application/sm

Encoding: -

Id: 70

Reference: [this document]

9. Acknowledgements

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. The authors would like to
thank Francesca Palombini for contributing to the discussion and giving helpful implementation input to the specification. We are grateful to Malisa Vucinic for providing many helpful comments.

10. References

10.1 Normative References

[I-D.ietf-jose-json-web-signature]

[I-D.ietf-jose-json-web-encryption]


10.2 Informative References

[I-D.seitz-ace-problem-description]

[I-D.ietf-ace-usecases]

[I-D.bormann-jose-cose]
Bormann, C., "Constrained Object Signing and Encryption
Appendix A. Which CoAP Header Fields and Options to Protect

In the case of CoAP Message Protection (Mode:COAP) as much as possible of the CoAP message is protected. However, 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.

A.1 CoAP Header Fields

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. Example 5 in Section 2 shows that the Code SHALL be integrity protected. The Version parameter SHALL neither be integrity protected nor encrypted (see Section 6).

A.2 CoAP Options
This section describes what options need to be integrity protected and encrypted. On a high level, all CoAP options must be encrypted by default, unless intended to be read by an intermediate node; and 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 for SSM and SEM.

A.2.1 Integrity Protection

As a general rule, CoAP options which are Safe-to-Forward SHALL be integrity protected, with the only exception being Enc and Sig, which are the security-providing options.

The Unsafe options are divided in two categories, those that are intended to change in a way that can be reconstructed by the server, and those which are not. The following options are of the latter kind and SHALL NOT be integrity protected: Max-Age, Observe, Proxy-Scheme. These options are intended to be changed by a proxy.

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. These options are Unsafe, but the Forward Proxy is intended to perform this precise operation and we can use this predictability to integrity protect the destination endpoint URI, even if the options where the information elements of the URI is located is changed by the Proxy.

This memo makes the full URI located in option 35 (Proxy-Uri) into a common denominator for the URI integrity, as described in the following. The following processing applies to a SSM, for SEM 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 SM Tag is calculated on the following message, 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 (no. 35), which is included in the integrity verification.

A.2.2 Encryption

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

- Max-Age, Observe: This information is intended to be read by a proxy.
- Enc, Sig: These are the security-providing options.
- Uri-Host, Uri-Port: This information can be inferred from destination IP address and port.
- Proxy-Uri, Proxy-Scheme: This information is intended to be read by a proxy.

In the case of a SEM, the Proxy-Uri MUST only contain Uri-Host and Uri-Port and MUST NOT contain Uri-Path and Uri-Query because the latter options are not intended to be revealed to a Forward Proxy.

A.2.3 Summary

Table 3 summarizes which options are encrypted and integrity protected, if present.

In a SSM, options marked with "a" and "b" are composed into a URI as described above and included as the Proxy-Uri option which is part of the SSM Body. In a SEM, options marked "a" are composed into a URI as described above and included as the Proxy-Uri option in the SEM Additional Authenticated Data.

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>E</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>If-Match</td>
<td>opaque</td>
<td>0-8</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td></td>
<td>Uri-Host</td>
<td>string</td>
<td>1-255</td>
<td>x</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>ETag</td>
<td>opaque</td>
<td>1-8</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>If-None-Match</td>
<td>empty</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>Observe</td>
<td>uint</td>
<td>0-3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3: Protected CoAP options in Mode=COAP.

Appendix B. JOSE Objects as Secure Messages

This section shows how to extend JWS and JWE to Secure Message formats (see Section 4.1). The use of compact serialization is assumed.

B.1 JWS as Secure Signed Message

The JOSE Header of JWS contains the mandatory parameter "alg", defined in Section 4.1.1 of [I-D.ietf-jose-json-web-signature], which corresponds to the parameter Algorithm of the Secure Message.

A JWS is a Secure Message if the JOSE Header includes

- the Parameter "kid" defined in Section 4.1.4 of [I-D.ietf-jose-json-web-signature];

- the new Parameter "seq" defined in B.3; and

- the new Parameter "mod" defined in B.4.

In case of JWS, a SSM with Detached Content consists of the JOSE Header and JWS Signature; i.e. no JWS Payload.

B.2 JWE as Secure Encrypted Message

In case of JWE, the SM Header parameters of a JWE consists of the JOSE Header Parameters and JWE Initialization Vector (IV).
The JOSE Header of JWE contains the mandatory parameter "enc", defined in Section 4.1.2 of [I-D.ietf-jose-json-web-encryption], which corresponds to the parameter Algorithm of the Secure Message. The JOSE Header also contains the mandatory parameter "alg", the key encryption algorithm, which in the current version of the draft is assumed to be equal to "dir" (constant). It is also assumed that plaintext compression (zip) is not used.

A JWE is a Secure Message if the IV contains the SM Sequence Number, and the JOSE Header includes

- the Parameter "kid" defined in Section 4.1.4 of [I-D.ietf-jose-json-web-signature]; and

- the new Parameter "mod" defined in B.4.

The IV also contain a Salt (see Section 6). For JWE it is mandatory to include the IV and hence the Salt is sent in each message.

In case of JWE, a SEM with Detached Content consists of JOSE Header, JWE Initialization Vector, JWE Ciphertext and JWE Authentication Tag; i.e. no JWE AAD.

B.3 "seq" (Sequence Number) Header Parameter

The Sequence Number SHALL be a 64-bit unsigned integer in hexadecimal representation. Only the significant bytes are sent (initial bytes with zeros are removed). The start sequence number SHALL be 0. For a given key, any Sequence Number MUST NOT be used more than once.

The parameter "seq" SHALL be marked as critical using the "crit" header parameter (see section 4.1.11 of [I-D.ietf-jose-json-web-signature]), meaning that if a receiver does not understand this parameter it must reject the message.

B.4 "mod" (Mode) Header Parameter

The Mode parameter SHALL be an 8-byte unsigned integer defining application specific message format, content and processing. The parameter "mod" SHALL be marked as critical. "mod":"0" indicates Mode:APPL which is defined in Section 5.2. "mod":"1" indicates Mode:COAP which is defined in Section 5.1.

B.4 The TID consists of the concatenation of SEQ and KID, in that order, formatted as in the JOSE. For "seq" the initial bytes with zeros are removed.
Appendix C. Compact Secure Message

For constrained environments it is important that the message expansion due to security overhead is kept at a minimum. As an attempt to assess what this minimum expansion could be, an optimized Secure Message format is defined, tailor-made for this setting. This is intended as a benchmark for generic content formats, to allow an informed decision about which Secure Message format to mandate in a future version of this draft.

C.1 CSM Format

This section defines a compact Secure Message format (see Section 4.1) called the Compact Secure Message (CSM) format, see Figure 4.

The CSM Header (see Section 4.1.1.) consists of 2 bytes of fixed length parameters and two variable length parameters, Key Identifier (KID) and Sequence Number (SEQ). The Header parameters are (compare Table 5):

- Mode (M). M=0 indicates Mode:APPL as defined in Section 5.2. M=1 indicates Mode:COAP as defined in Section 5.1. M=2 and M=3 are reserved for future use.
- Algorithm (ALG). This parameter consists of an encoding of the ciphersuite used in the Secure Message. The encoding is TBD.
- KID Length (KL). This parameter consist of a length indication of the header parameter Key Identifier. The actual length of KID is KL + 1 bytes.
- SEQ Length (SL). This parameter consist of a length indication of the header parameter Sequence Number. The actual length of
SEQ is SL + 1 bytes.

- **Key Identifier (KID).** This parameter identifies the key(s) used to protect the Secure Message. Only the significant bytes are sent (initial bytes with zeros are removed).

- **Sequence Number (SEQ).** This parameter consists of the sequence number used by the sender of the Secure Message. Only the significant bytes are sent (initial bytes with zeros are removed).

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameter</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Mode</td>
<td>2 bits</td>
</tr>
<tr>
<td>ALG</td>
<td>Algorithm</td>
<td>6 bits</td>
</tr>
<tr>
<td>KL</td>
<td>Key Identifier Length</td>
<td>5 bits</td>
</tr>
<tr>
<td>SL</td>
<td>Sequence Number Length</td>
<td>3 bits</td>
</tr>
<tr>
<td>KID</td>
<td>Key Identifier</td>
<td>KL + 1: 1-32 bytes</td>
</tr>
<tr>
<td>SEQ</td>
<td>Sequence Number</td>
<td>SL + 1: 1-8 bytes</td>
</tr>
</tbody>
</table>

Table 5: CSM Header Parameters.
The minimum CSM Header is 4 bytes.

The TID consists of the concatenation of SEQ and KID, in that order, formatted as in the CSM format (initial bytes with zeros are removed).

The content of CSM Body depends on whether it is a SSM or a SEM (see Section 4.1.2) which is determined by the Algorithm. This version of the draft focuses on Secure Message with Detached Content. Hence, the SSM Body is empty and the SEM Body consists of the Ciphertext. In the former case, the length of the CSM Body is 0. In the latter case, the length of the CSM Body equals the sum of the lengths of the present CoAP options marked encrypted in Table 3 and the length of the payload of the unprotected CoAP message.

The CSM Tag contains the MAC/Signature as determined from the Algorithm. The length is determined by ALG.
C.2 Comparison of Secure Message sizes

This section gives some examples of overhead incurred with JOSE, the current proposal for COSE at the time of writing (00-draft), and CSM. The goal is not to give exact measurements, but to help the reader appreciate the rough order of magnitude of the overhead involved. COSE seems to be the most promising approach and CSM should be viewed as an attempt to define a lower bound for COSE.

The comparison is complicated further by the fact that algorithms suitable for constrained environments are not supported by JOSE, and thereby not by COSE. This comparison does not consider the ciphertext or signed payload expansion due to Base64url encoding in JWS/JWE. This would increase the overhead of JWS and JWE even more.

The size of the header is shown separately from the size of the authentication tag, since JWS/JWE has no provisions for truncating it, a feature that could easily be added to the JOSE specifications. For CSM the encoding of certain additional algorithms is assumed and this could also easily be added to COSE. An 8-byte kid is used throughout all examples. Finally compact serialization for both JWS and JWE is assumed.

SSM uses HMAC-SHA256, with truncation to 16 bytes.

For JWS the following header is used:

{"alg":"HS256", "kid":"a1534e3c5fdc09bd", "seq":"00000142", "mod":"0"}

which encodes to a size of 90 bytes in Base64url, and the 32 bytes of HS256 MAC encode to 43 bytes. The concatenation marks add 2 bytes to that in the total overhead.

The same header in COSE, representing the "kid" as bytes (not as string) and the "seq" as positive integer encodes to a size of 35 bytes, and the MAC would add to 32 bytes to that. Note that encoding the header and the MAC together incurs an additional overhead of 3 bytes.

For CSM the same header is represented by 12 bytes. The MAC could in this case safely be truncated to 16 bytes, and a corresponding algorithm identifier would need to be defined in the list of supported algorithms.

Table 6 summarizes these results.
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>MAC</th>
<th>Total Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>JWS</td>
<td>90 B</td>
<td>43 B</td>
<td>135 bytes</td>
</tr>
<tr>
<td>COSE</td>
<td>35 B</td>
<td>32 B</td>
<td>70 bytes</td>
</tr>
<tr>
<td>CSM</td>
<td>12 B</td>
<td>16 B</td>
<td>28 bytes</td>
</tr>
</tbody>
</table>

Table 6: Comparison of JWS, COSE, and CSM

For SEM the use of AES-128-CCM-8 would be ideal, but since this is not supported by JOSE, AES-128-GCM is used there instead.

For JWE 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 IV in the CSM format, but also that the JWE and COSE formats can omit the sequence number.

The JWE header

{"alg":"dir", "kid":"a1534e3c5fdc09bd", "enc":"A128GCM", "mod":"0"}

encodes to a size of 86 bytes in Base64url, while the necessary 12 byte IV for GCM mode is expanded to 16 bytes by encoding. The 16 bytes of the authentication tag expand to 22 bytes. The concatenation marks add 3 bytes to the total overhead.

In COSE the same header encodes to 40 bytes and the IV and authentication tag could be represented as 12 and 16 bytes respectively. Note that encoding the header, the IV and the authentication tag together incurs an additional overhead of 2 bytes.

For CSM this tests uses CCM mode instead of GCM. CCM requires a 16 byte IV, but is better suited for constrained devices, and for CSM there is no impact since the IV can be deduced from the sequence number and a previously agreed upon Salt. The corresponding header for AES-128-CCM-8, including the 8 byte sequence number, is represented by 12 bytes.

Table 7 summarizes these results.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>IV</th>
<th>Tag</th>
<th>Total Overhead</th>
</tr>
</thead>
</table>

Appendix D. Examples

This section gives examples of how to use the new options and message formats defined in this memo.

D.1 CoAP Message Protection

This section illustrates the Secure Message Mode:COAP. 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 a SM Mode:COAP object (Sig or Enc option is set) and thus forwards the message as it cannot serve a cached response.

D.1.1 Integrity protection of CoAP Message

Here is an example of a PUT request/response message exchange passing an intermediate node protected with the Sig option. The example illustrates a client opening a lock and getting a confirmation that the lock is opened. Code, Uri-Path and Payload are integrity protected (see Appendix A).

Client  Proxy  Server

|      |      |      |
|      |      |      |
|      |      |      |
| +---->|      |      |
| PUT  |      |      |
| Code: 0.03 (PUT) Token: 0x8c Uri-Path: lock Sig: SSM {"mod":"1","seq":"00000142", "kid":"a1534e3c5fcd09bd", ...} Payload: 1 |
|      |      |      |
| +----->|      |      |
| PUT  |      |      |
| Code: 0.03 (PUT) Token: 0x7b Uri-Path: lock Sig: SSM {"mod":"1","seq":"00000142", ...} |
"kid":"a1534e3c5f4c0b9d", ...

Payload: 1

Figure 8: CoAP PUT protected with Sig/SSM (Mode: COAP)

The Key Identifier is a hint to the receiver 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, and since Mode is COAP, the client can additionally verify the freshness of the response, i.e. that the response message is generated as an answer to the received request message (see Section 5.3).

The SSM also contains the Tag as specified in the Algorithm (not shown).

This example deviates from encryption (SEM) by default (see Section 6) just to illustrate the Sig option. If there is no compelling reason why the CoAP message should be in plaintext, then the Enc option must be used.

D.1.2 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 Appendix A).

Client  Proxy  Server

Figure 9: CoAP GET protected with Enc/SEM (Mode: COAP).

The bracket [ ... ] indicates encrypted data.

Since the request message (GET) does not support payload, the SEM is carried in the Enc option. Since the response message (Content) supports payload, the Enc option is empty and the SEM is carried in the payload.

The Key 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 since Mode: COAP the client can additionally verify the freshness of the response, i.e. that the response message is generated as an answer to the received request message (see Section 5.3).
The SEM also contains the Tag as specified by the Algorithm (not shown).

D.2 Application Layer Protection

This section gives examples that illustrate Secure Message Mode:APPL. This mode assumes that only the intended receiver(s) has the relevant security context related to the resource.

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 SEM object, and serve this response to another client request, such that both clients can verify integrity and non-replay.

Client1 Proxy Server

+-----+ Code: 0.01 (GET) Token: 0x83 Proxy-Uri: example.com/temp
GET

+-----+ Code: 0.01 (GET) Token: 0xbe
GET Uri-Host: example.com Uri-Path: temp

<----- Code: 2.05 (Content) Token: 0xbe
2.05 Payload: SEM {"mod":"0","seq":"000015b7", "kid":"c09bda155fd34e3c", ["471 F"], ...}

<----- Code: 2.05 (Content) Token: 0x83
2.05 Payload: SEM {"mod":"0","seq":"000015b7", "kid":"c09bda155fd34e3c", ["471 F"], ...}

Client2

+-----+ Code: 0.01 (GET)
D.2.2 Publish-Subscribe

This example outlines a publish-subscribe setting where the payload is integrity and replay protected end-to-end between Publisher and Subscriber. The example illustrates a subscription registration and a new 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 Mode:COAP), but cannot manipulate the Sequence Number without being noticed.

Sub- Publisher Broker

```plaintext
+------  Code: 0.01 (GET)
| GET    Token: 0x72
| Uri-Path: ps
| Uri-Path: birch-pollen
| Observe: 0 (register)

<------  Code: 2.05 (Content)
2.05    Token: 0x72
Observe: 1
Payload: SSM {"mod":"0","seq":"000015b7","kid":"c09bda155fd34e3c","["270"], ...}
```

```plaintext
<------  Code: 0.03 (PUT)
PUT     Token: 0x1f
Uri-Path: ps
Uri-Path: birch-pollen
Payload: SSM {"mod":"0","seq":"000015b8","["471 F"], ...}
```
This example deviates from encryption (SEM) by default (see Section 6) just to illustrate the SSM in Mode:APPL. If there is no compelling reason why the payload should be in plaintext, then SEM 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 Section 4.4-4.5 of [I-D.seitz-ace-problem-description].

Here, C is clearly not trusted with modifying the information, but may need to be involved with 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 is a natural candidate (cf. "Access Tokens").

This example considers the authorization information to be encapsulated in a SEM Mode:APPL object, generated by AS. How C accesses the SSM 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 SEM by doing PUT to /authorization. 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 12.

Authz       Resource
Server       Client       Server

Figure 11: Publish-subscribe protected with SSM (Mode:APPL)
Client access Access Token:

```
SEM { "mod":"0", "seq":"00000142", "kid":"c09bda1534e3c5f6c09bd", ... }
```

-----

Code: 0.03 (PUT)

PUT

Token: 0xac

Uri-Path: authorization

Payload:

```
SEM { "mod":"0", "seq":"00000142", "kid":"c09bda1534e3c5f6c09bd", ... }
```

Figure 12: Protected Transfer of Access Token = SEM (Mode:APPL)

Authors’ Addresses

Goeran Selander
Ericsson
Farogatan 6
16480 Kista
SWEDEN
EMail: goran.selander@ericsson.com

John Mattsson
Ericsson
Farogatan 6
16480 Kista
SWEDEN
EMail: john.mattsson@ericsson.com

Ludwig Seitz
SICS Swedish ICT AB
Scheelelaven 17
22370 Lund
SWEDEN
CoAP Communication with Alternative Transports
draft-silverajan-core-coap-alternative-transports-07

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 understanding how CoAP can operate with other transports, such as the need for M2M communication using non-IP networks, improved transport level end-to-end reliability and security, NAT and firewall traversal issues, and mechanisms possibly incurring a lower overhead to CoAP/HTTP translation gateways. This draft examines the requirements for conveying CoAP messages to end points over such alternative transports. It also provides a new URI format for representing CoAP resources over alternative transports.

Status of This Memo

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

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

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

This Internet-Draft will expire on June 20, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.
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.

As the Internet continues taking shape 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, the CoAP endpoint can reside in a non-IP network, be separated from its peer by NATs and firewalls or simply has no possibility to communicate over UDP. Consequently in addition to UDP, alternative transport channels for conveying CoAP messages could be considered.

Extending CoAP over alternative transports allows 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. 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 Requests and Responses 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 Request and Response 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. The Open Mobile Alliance (OMA) specifies both UDP and SMS as transports for M2M communication in cellular networks. The OMA Lightweight M2M protocol being drafted uses CoAP, and as transports, specifies both UDP binding as well as Short Message Service (SMS) bindings [OMALWM2M] for the same reason. For securing end-to-end communication between Mobile Stations, the use of DTLS-encoded CoAP messages over SMS is also being proposed [I-D.fossati-dtls-over-gsm-sms].

2.2. Use of WebSockets

The WebSocket protocol is being proposed as a transport channel between WebSocket enabled CoAP end-points on the Internet [I-D.savolainen-core-coap-websockets]. This is particularly useful as a means for web browsers, especially in smart devices, to allow embedded client side scripts to create new WebSocket connections to various WebSocket-enabled servers, through which CoAP Request and Response messages can be exchanged. This also allows a browser containing an embedded CoAP server to behave as a WebSocket client by opening a connection to a WebSocket enabled CoAP Mirror Server [I-D.vial-core-mirror-server] to register and update its resources.

2.3. Use of P2P Overlays

[I-D.jimenez-p2psip-coap-reload] specifies how CoAP nodes can use a peer-to-peer overlay network called RELOAD, as a resource caching facility for storing wireless sensor data. When a CoAP node registers its resources with a RELOAD Proxy Node (PN), the node computes a hash value from the CoAP URI and stores it as a structure together with the PN’s Node ID as well as the resources. Resource retrieval by CoAP nodes is accomplished by computing the hash key over the Request URI, opening a connection to the overlay and using its message routing system to contact the CoAP server via its PN.

2.4. Use of TCP and TLS

Using TCP to facilitate the traversal of CoAP Request and Response messages [I-D.bormann-core-coap-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-functionality [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.

2.5. Others

CoAP could in addition be extended atop other transport channels, such as:

1. The transportation of CoAP messages in Delay-Tolerant Networks [RFC4838], using the Bundle Protocol [RFC5050] for reaching sensors in extremely challenging environments such as acoustic, underwater and deep space networks.

2. Any type of non-IP networks supporting constrained nodes and low-energy sensors, such as Bluetooth and Bluetooth Low Energy (either through L2CAP or with GATT) [BTCorev4.1], ZigBee, Z-Wave, 1-Wire, DASH7 and so on.

3. Instant Messaging and Social Networking channels, such as Jabber and Twitter.

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.
Nodes falling under Type T0 possess the capability of exactly 1 type of transport channel for CoAP, at all times. These include both active and sleepy nodes, which may choose to perform duty cycling for power saving.

Type T1 nodes possess multiple different transports, and can retrieve or expose CoAP resources over any or all of these transports. However, not all transports are constantly active and certain transport channels and interfaces could be kept in a mostly-off state for energy-efficiency, such as when using CoAP over SMS (refer to section 2.1).

Type T2 nodes possess more than 1 transport, and multiple transports are simultaneously active at all times. CoAP proxy nodes which allow CoAP endpoints from disparate transports to communicate with each other, are a good example of this.

4. CoAP Alternative Transport URI

Based on the usage scenarios as well as the transport classes presented in the preceding sections, this section discusses the formulation of a new URI for representing CoAP resources over alternative transports.

CoAP is logically divided into 2 sublayers, whereby a request/response 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

\[
\text{URI} = \text{scheme ":" } \text{//" authority path-abempty ["?"query ]}
\]
a simple example CoAP URI, "coap://server.example.com/sensors/ temperature" is interpreted as follows:

\[
\text{coap :}// \text{server.example.com} /sensors/temperature
\]

protocol | endpoint | parameterised
--- | --- | ---
identifier | identifier | resource
identifier

Figure 1: The CoAP URI format

The resource path is explicitly expressed, and the endpoint identifier, which contains the host address at the network-level is also directly bound to the scheme name containing the application-level protocol identifier. The choice of a specific transport for a scheme, however, cannot be embedded with a URI, but is defined by convention or standardisation of the protocol using the scheme. As examples, [RFC5092] defines the 'imap' scheme for the IMAP protocol over TCP, while [RFC2818] requires that the 'https' protocol identifier be used to differentiate using HTTP over TLS instead of TCP.

4.1. Design Considerations

Several ways of formulating a URI which express an alternative transport binding to CoAP, can be envisioned. When such a URI is provided from an end-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. 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. Also, the generic syntax for a URI is 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. 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.

3. 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
o coap+sms://0015105550101/sensors/temperature for using CoAP over SMS with the endpoint identifier being a telephone subscriber number

o coaps+sms://0015105550101/sensors/temperature for using CoAP over DTLS over SMS with the endpoint identifier being a telephone subscriber number

o coap+ws://www.example.com/sensors/temperature for using CoAP over WebSockets

o coap+wss://www.example.com/sensors/temperature for using CoAP over secure WebSockets (WebSockets using TLS)

A URI of this format to distinguish transport types is simple to understand and not dissimilar to the CoAP URI format. As the usage of each alternative transport results in an entirely new scheme, IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [I-D.ietf-appsawg-uri-scheme-reg], particularly where permanent URI scheme registration is concerned.

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 this document. 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. Note that Properties 1, 2, and 3 are related.

Property 1: Uniqueness of an end-point identifier.

Transport protocols providing non-unique end-point IDs for nodes may only convey a subset of the CoAP functionality. Such nodes may only serve as CoAP servers that announce data at specific intervals to a pre-specified end point, or to a shared medium.
Property 2: Unidirectional or bidirectional CoAP communication support.

This refers to the ability of the CoAP end-point to use a single transport channel for both request and response messages. Depending on the scenario, having a unidirectional transport layer would mean the CoAP end-point might utilise it only for outgoing data or incoming data. Should both functionalities be needed, 2 unidirectional transport channels would be necessary.

Property 3: 1:N communication support.

This refers to the ability of the transport protocol to support broadcast and multicast communication. CoAP's request/response behaviour depends on unicast messaging. Group communication in CoAP is bound to using multicasting. Therefore 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 4: Transport-level reliability.

This refers to the ability of the transport protocol to provide a guarantee of 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 5: 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, although techniques exist in allowing binary payloads to be transferred over text-based transport protocols such as base-64 encoding. A fuller discussion about performing CoAP message encoding for SMS can be found in Appendix A.5 of [I-D.bormann-coap-misc].

Property 6: 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 7: 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 8: 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 9: 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 10: 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

While no new security risks are envisaged simply from the introduction of support for alternative transports, end-applications and CoAP implementations should take note if certain transports require privacy trade-offs that may arise if identifiers such as MAC addresses or phone numbers are made public in addition to FQDNs.

8. Acknowledgements

Feedback, ideas and ongoing discussions with Klaus Hartke, Martin Thomson, Mark Nottingham, Dave Thaler, Graham Klyne, Carsten Bormann, Markus Becker and Golnaz Karbaschi provided useful insights and ideas for this work.

9. References

9.1. Normative References


9.2. Informative References


[I-D.vial-core-mirror-server]

[OMALWM2M]

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

[RFC2609]

[RFC2818]

[RFC4838]

[RFC5050]

[RFC5092]

[RFC6455]

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

[RFC6733]

[RFC7390]

[WWWArchv1]
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-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.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:

```
coop-at://srv-_coap._tcp.example.com/sensors/temperature
```

in which the "srv" prefix is used to indicate that a DNS SRV lookup should be used for _coap._tcp.example.com, where usage of CoAP over TCP is specified for example.com, and is eventually resolved to a numerical IPv4 or IPv6 address.

A.2. Making CoAP Resources Available over Multiple Transports

The CoAP URI used thus far is as follows:

```
URI         = scheme ":" hier-part [ "?" query ]
hier-part   = "//" authority path-abempty
```

A new URI format could be introduced, that does not possess an "authority" component, and instead defining "hier-part" to instead use another component, "path-rootless", as specified by RFC3986 [RFC3986]. The partial ABNF format of this URI would then be:

```
URI         = scheme ":" hier-part [ "?" query ]
hier-part   = path-rootless
path-rootless = segment-nz *( "/" segment )
```

The full syntax of "path-rootless" is described in [RFC3986]. A generic URI defined this way would conform to the syntax of [RFC3986], while the path component can be treated as an opaque string to indicate transport types, endpoints as well as paths to CoAP resources. A single scheme can similarly be used.

A constrained node that is capable of communicating over several types of transports (such as UDP, TCP and SMS) would be able to convey a single CoAP resource over multiple transports. This is also beneficial for nodes performing caching and proxying from one type of transport to another.
Requesting and retrieving the same CoAP resource representation over multiple transports could be rendered possible by prefixing the transport type and endpoint identifier information to the CoAP URI. This would result in the following example representation:

```
coap-at:tcp://example.com?coap://example.com/sensors/temperature
```

![Figure 2: Prefixing a CoAP URI with TCP transport](image)

Such a representation would result in the URI being decomposed into its constituent components, with the CoAP resource residing within the query component as follows:

- **Scheme**: coap-at
- **Path**: tcp://example.com
- **Query**: coap://example.com/sensors/temperature

The same CoAP resource, if requested over a WebSocket transport, would result the following URI:

```
coap-at:ws://example.com/endpoint?coap://example.com/sensors/temperature
```

![Figure 3: Prefixing a CoAP URI with WebSocket transport](image)

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


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
Abstract

CoAP has been standardised as an application level REST-based protocol. This document introduces a way for CoAP clients and servers to interact with resources by agreeing upon alternate locations as well as transport and protocol configurations.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

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

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

The Constrained Application Protocol (CoAP) [RFC7252] provides a lightweight request-response messaging mechanism for retrieving and manipulating resources identified by Uniform Resource Identifiers (URIs). However, URIs have a twofold purpose in CoAP: In addition to identifying resources, URIs are also used as locators for origin servers, proxies, and endpoints delivering resource representations to clients. 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.

This draft proposes a new link format attribute as well as a new link relation type that together enable an origin server to serve a resource from other protocol configurations or endpoints. CoAP clients then interact with an origin server’s CoRE resource discovery interface to obtain a set of links describing alternate locations of resources.

This is 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.

2. New Link Attribute and Relation types

A CoAP server wishing to allow interactions with resources from multiple locations or transports can do so by specifying the Transport Type "tt" link attribute, which is an opaque string. Multiple transport types can be included in the value of this parameter, each separated by a space. In such cases, transport types appear in a prioritised list, with the most preferred transport type by the CoAP server specified first and the lowest priority transport type last.

At the same time, each transport type supported by the server is also described with an "altloc" link relation type. The "altloc" relation type specifies a URI (containing the URI scheme, authority and optionally path) providing an alternate endpoint location up to but not including the resource path of a representation.

Both "tt" and "altloc" are optional CoAP features. If supported, they occur at the granularity level of an origin server, i.e. 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?)

3. 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=sm

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.
4. IANA Considerations

New link attributes and link relations need to be registered.

5. Security Considerations

 Probably lots. (TBD)

6. Acknowledgements

Thanks to Klaus Hartke for comments and reviewing this draft, and Teemu Savolainen for initial discussions about protocol negotations and lifetime values.

7. References

7.1. Normative References


7.2. Informative References

[I-D.silverajan-core-coap-alternative-transports]


[WWWArchv1]

Author’s Address

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi
A TCP and TLS Transport for the Constrained Application Protocol (CoAP)
draft-tschofenig-core-coap-tcp-tls-02.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 September 10, 2015.

Copyright Notice

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

The Internet protocol stack is organized in layers, namely link layer, internet layer, transport layer, and the application layer ([RFC1122]).

IP emerged as the waist of the hour glass and supports a variety of link layers and new link layer technologies can be added in the future, without affecting IP.

Combined with the end-to-end principle, the hour glass indicates the level of protocol understanding that intermediaries need to have in order to forward IP packets between a sender and a receiver (absent any specific application layer entities, such as proxies or caches). Having IP as the waist means that anyone can extend the layers above the network layer in the way they want to communicate end-to-end, including defining new transport layer protocols.
Unfortunately, some network deployments depart from this architecture. 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 even supported by firewalls. Middleboxes that are unaware of the IoT can make the use of UDP brittle.

As a separate consideration, some environments benefit from the more advanced congestion control and flow control capabilities provided by TCP. For instance, CoAP back-end processors in a cloud environment may want to connect between each other via TCP instead of UDP; a TCP-to-UDP gateway can be used at the cloud boundary to talk to the UDP-based IoT.

To make both IoT devices and their associated back-end processors 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).
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 supported 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).

As a result of removing the message layer in CoAP over TCP, the only supported message type from the ones CoAP over UDP provides is the
NON type. A response is sent back as defined in [RFC7252], as illustrated in Figure 3 (derived from [RFC7252], Figure 6).

Client

NON [-------]
GET /temperature
(Token 0x74)

Server

NON [-------]
2.05 Content
(Token 0x74)
"22.5 C"

<------------------+

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.

<table>
<thead>
<tr>
<th>Ver</th>
<th>T</th>
<th>TKL</th>
<th>Code</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Token (if any, TKL bytes) ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Options (if any) ...</td>
</tr>
<tr>
<td>-----</td>
<td>-------------------------------</td>
</tr>
<tr>
<td></td>
<td>Payload (if any) ...</td>
</tr>
</tbody>
</table>

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 the 2-byte shim header carrying length information prepending the CoAP message header.
The ‘Message Length’ field is a 16-bit unsigned integer in network byte order. It provides the length of the subsequent CoAP message (including the CoAP header but excluding this message length field) in bytes. T is always the code for NON (1). 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 of subsequent 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. 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.

5.1. coap+tcp URI scheme

coop-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.)
5.2. 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? --cabo]]

6. 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].

7. IANA Considerations

7.1. 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>
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 7.3 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 7.3 of [RFCthis])

7.2. 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].

7.3. 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]

8. Acknowledgements

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Michael Koster, Matthias Kovatsch, Szymon Sasin, and Zach Shelby for their feedback.

9. References

9.1. Normative References

[I-D.ietf-dice-profile]


9.2. Informative References

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",
draft-ietf-core-block-16 (work in progress), October 2014.

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

[RFC1122] Braden, R., "Requirements for Internet Hosts -

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

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
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).

Status of This Memo

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

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

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

This Internet-Draft will expire on August 7, 2015.
Copyright Notice

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

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

Table of Contents

1. Introduction ........................................ 3
   1.1. Design considerations .............................. 4
   1.2. Terminology ........................................ 5
       1.2.1. Tree Diagrams ................................ 5
2. CoMI Architecture .................................... 6
   2.1. RESTCONF/YANG Architecture ...................... 9
3. CoAP Interface ....................................... 10
4. MG Function Set ...................................... 11
   4.1. Data Retrieval ..................................... 11
       4.1.1. GET ............................................. 12
       4.1.2. Mapping of the ‘select’ Parameter .............. 12
       4.1.3. Retrieval Examples ........................... 13
   4.2. Data Editing ........................................ 23
       4.2.1. POST ............................................. 23
       4.2.2. PUT ............................................. 23
       4.2.3. DELETE .......................................... 24
   4.3. Notify functions ................................... 24
   4.4. Module Discovery .................................. 25
   4.5. Error Return Codes ................................ 27
5. Mapping YANG to CoMI payload ......................... 28
   5.1. YANG Hash Generation .............................. 29
   5.2. Re-Hash Procedure ................................ 29
   5.3. ietf-yang-hash YANG Module ...................... 30
       5.3.1. YANG Re-Hash Example ........................ 32
   5.4. YANG Hash in URL .................................. 33
6. Mapping YANG to CBOR ................................ 33
   6.1. High level encoding ................................ 33
   6.2. Conversion from YANG datatypes to CBOR datatypes 34
7. Error Handling ........................................ 35
8. Security Considerations .............................. 36
9. IANA Considerations .................................. 37
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]. RESTCONF allows access to data resources contained in NETCONF [RFC6241] datastores. RESTCONF messages can be encoded in XML [XML] or JSON. 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.

A large amount of Management Information Base (MIB) [RFC3418] specifications already exist for monitoring purposes. This data can be accessed in RESTCONF if the server converts the SMIv2 modules to YANG, using the mapping rules defined in [RFC6643].

The CoRE Management Interface (CoMI) is intended to work on standardized data-sets in a stateless client-server fashion. The RESTCONF protocol is adapted and optimized for use in constrained environments, using CoAP instead of HTTP. Standardized data sets promote interoperability between small devices and applications from different manufacturers. Stateless communication is encouraged to keep communications simple and the amount of state information small in line with the design objectives of 6lowpan [RFC4944] [RFC6775], RPL [RFC6650], and CoAP [RFC7252].

RESTCONF uses the HTTP methods HEAD, OPTIONS, and PATCH, which are not available in CoAP. HTTP uses TCP which is not recommended for CoAP. The transport protocols available to CoAP are much better suited for constrained networks.
TODO: Introduce CoAP Patch options to allow modification to subsets of resource.

CoMI is low resource oriented, uses CoAP, and only supports the methods GET, PUT, POST and DELETE. The payload of CoMI is encoded in CBOR [RFC7049] which is automatically generated from JSON [JSON]. 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 minimise 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. 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 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 CoAP based 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 end goal of CoMI is to provide information exchange over the CoAP transport protocol 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 detailed comparison between CoMI and LWM2M management can be found in Appendix C. CoMI supports MIB modules which have been translated 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.

CoMI uses a simple URI to access the management object resources. Complexity introduced by instance selection, or multiple object specification is expressed with uri-query attributes. The choice for uri-query attributes makes the URI structure less context dependent.
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, datastore, 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, datastore resource, edit operation, query parameter, target resource, and unified datastore.

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 a request target URI, error-path or other string, then the lowercase hexadecimal representation is used. Leading zeros are used so the value uses 8 hex characters.

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 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 signature of the CoAP request in the server 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
analysing 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, GET, DELETE, and POST. New variables can be created with POST when they exist in the YANG specification. The Observe option can be 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 module name, the smi context, 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 to 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 datastore (see Section 2.1). CoMI does not see the different variable store types. The Variable store contains instances of the YANG specification. 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 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 datastore 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 datastore, to allow
stateless editing of configuration variables and the notification of
operational variables.

The child schema nodes of the unified datastore 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 the 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 datastore itself (to
retrieve the entire datastore or create a top-level data resource
instance). CoMI uses a 30-bit YANG hash value (based on the YANG
data node path identifier strings) to identify schema nodes in the target resource URI and in the payload.

Any message payload data is relative to the node specified in the target resource URI in a request message. CoMI message payloads are based on the JSON encoding of a RESTCONF message payload. The JSON identifier names are first converted to their 30-bit YANG hash values and then the payload is converted to CBOR.

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 4.4).

The mg resource has three sub-resources accessible with the paths:

/mg: YANG-based data with path "/mg" and using CBOR content encoding format. This path represents a datastore 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 5.4).

/mg/mod.uri: URI indicating 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.

/mg/yang.hash: URI indicating the location of the server YANG hash information if any objects needed to be re-hashed by the server. It has path "/mg/yang.hash" and is encoded in CBOR format. The "ietf-yang-hash" module of Section 5.3 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.

The mapping of YANG data nodes 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]:

<table>
<thead>
<tr>
<th>name</th>
<th>path</th>
<th>rt</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>/mg</td>
<td>core.mg</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>/mg</td>
<td>core.mg.data</td>
<td>application/cbor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module Set</td>
<td>/mg/mod.uri</td>
<td>core.mg.moduri</td>
<td>application/cbor</td>
</tr>
<tr>
<td>URI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YANG Hash</td>
<td>/mg/yang.hash</td>
<td>core.mg.yang-hash</td>
<td>application/cbor</td>
</tr>
<tr>
<td>Info</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieve the datastore resource or a data resource</td>
</tr>
<tr>
<td>POST</td>
<td>Create a data resource</td>
</tr>
<tr>
<td>PUT</td>
<td>Create or replace a data resource</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete a data resource</td>
</tr>
</tbody>
</table>

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. 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.

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.

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keys</td>
<td>Request to select instances of a YANG definition</td>
</tr>
<tr>
<td>select</td>
<td>Request selected sub-trees from the target resource</td>
</tr>
</tbody>
</table>

The "keys" parameter is used to specify a specific instance of the resource. When keys is not specified, all instances are returned. When no or one instance of the resource exists, the keys parameter is not needed.

4.1.2. Mapping of the ‘select’ Parameter

ANUJ TODO: Add more details based on the RESTCONF ‘select’ parameter. We need to add information about how this parameter is encoded. There should be an error notification when filtering fails.

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.

4.1.3. Retrieval Examples

The examples in this section use a JSON payload with one or more entries describing the pair (identifier, value). CoMI transports the CBOR format to transport the equivalent contents. The CBOR syntax of the payloads is specified in Section 5.

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.

In all examples: (1) the payload is expressed in JSON, although the operational payload is specified to be in CBOR, (2) the path is expressed in readable names although the transported path is...
expressed 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), and (3) only one instance is associated with the resource.

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:
/sys/system-state/sys:clock/sys:current-datetime

The 30 bit murmur3 hash value is calculated on this string (0x15370408 and VNwQI). The request using this hash value is shown below:

REQ: GET example.com/mg/VNwQI

RES: 2.05 Content (Content-Format: application/cbor)
{
  0x15370408 : "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. 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:

REQ: GET example.com/mg/system-state/clock
(Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
  "clock" : {
    "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):

/sys:system-state/sys:clock (0x2eb2fa3b and usvo7)
/sys:system-state/sys:clock/sys:current-datetime (0x15370408)
/sys:system-state/sys:clock/sys:boot-datetime (0x1fa25361)

The request using the hash values is shown below:

REQ: GET example.com/mg/usvo7
(Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
0x2eb2fa3b : {
0x15370408 : "2014-10-26T12:16:51Z",
0x1fa25361 : "2014-10-21T03:00:00Z"
}
}

4.1.3.2. Multiple instance retrieval

The specified 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 (identifier, value) pair. For example, the GET of the /interfaces/interface/ipv6/neighbor instance identified with interface index "eth0" [RFC7223], sent by the client, results in the following returned payload sent by the managed entity:
REQ: GET example.com/mg/interfaces/interface/ipv6/neighbor?keys=eth0
(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/ietf:interface/ipv6/ipv6:neighbor (0x2354bc49 and jVLxJ)
/ietf:interfaces/ietf:interface/ipv6/ipv6:neighbor/ipv4:ip (0x20b8907e and guJB_)
/ietf:interfaces/ietf:interface/ipv6/ipv6:neighbor/ipv6:link-layer-address (0x16f47fd8)

The request using the hash values is shown below:
REQ: GET example.com/mg/jVLxJ?keys=eth0
(Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
  0x2354bc49 : [
    {
      0x20b8907e : "fe80::200:f8ff:fe21:67cf",
      0x16f47fd8 : "00:00::10:01:23:45"
    },
    {
      0x20b8907e : "fe80::200:f8ff:fe21:6708",
      0x16f47fd8 : "00:00::10:54:32:10"
    },
    {
      0x20b8907e : "fe80::200:f8ff:fe21:88ee",
      0x16f47fd8 : "00:00::10:98:76:54"
    }
  ]
}

4.1.3.3. Access to MIB Data

The YANG translation of the SMI specifying the
ipNetToMediaTable 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:
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):
The following example shows a request for the entire ipNetToPhysicalTable. Since all the instances are requested, no "keys" query parameter is needed.
REQ: GET example.com/mg/IP-MIB/ipNetToPhysicalTable

REQ: GET example.com/mg/wt7w_

RES: 2.05 Content (Content-Format: application/cbor)
{ 0x30b7bc3f : {
   0x1067f289 : [
      0x00d38564 : 1,
      0x2745e222 : "ipv4",
      0x387804eb : "10.0.0.51",
      0x1a51514a : "00:00:10:01:23:45",
      0x03f95578 : "2333943",
      0x24ade115 : "static",
      0x09e640ef : "reachable",
      0x3b5c1ab6 : "active"
   },
   0x00d38564 : 1,
   0x2745e222 : "ipv4",
   0x387804eb : "9.2.3.4",
   0x1a51514a : "00:00:10:54:32:10",
   0x03f95578 : "2329836",
   0x24ade115 : "dynamic",
   0x09e640ef : "unknown",
   0x3b5c1ab6 : "active"
  ]
}

4.1.3.4. The 'keys' Query Parameter

There is a mandatory 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. Any 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).
REQ: GET example.com/mg/IP-MIB/ipNetToPhysicalTable/
ipNetToPhysicalEntry?keys=1,ipv4,10.0.0.51

REQ: GET example.com/mg/QZ/KJ?keys=1,ipv4,10.0.0.51

RES: 2.05 Content (Content-Format: application/cbor)
{
  0x1067f289 : [
    {
      0x00d38564 : 1,
      0x2745e222 : "ipv4",
      0x387804eb : "10.0.0.51",
      0x1a51514a : "00:00:10:01:23:45",
      0x03f95578 : "2333943",
      0x24ade115 : "static",
      0x09e640ef : "reachable",
      0x3b5c1ab6 : "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:A/foo:B/foo:col1
The YANG has value for this identifier string 0xa9abdcca and pq9zK).

The following string represents the RESTCONF target resource URI expression for the "col1" leaf for the key values "top", 17, and "group1":

/restconf/data/foo-mod:A=top,17/B=group1/col1

The following string represents the CoMI target resource identifier for the same instance of the "col1" leaf:

/mg/pq9zK?keys=top,17,group1

4.2. Data Editing

CoMI allows datastore contents to be created, modified and deleted using CoAP methods.

TODO: Data-editing is an optional feature. A server can choose to only support YANG modules with read-only objects.

4.2.1. 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.

TODO: CoMI does not support user-ordered lists in YANG.

4.2.2. 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.

TODO: Define where PATCH is needed.

4.2.3. 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 PUT method.

4.3. Notify functions

Notification by the server to a selection of clients when the value of a management object changes is an essential function for the management of servers. CoMI allows to do a notifications on all variables in the datastore.

Notification of object changes is supported with the CoAP Observe [I-D.ietf-core-observe] function. The client subscribes to the object by sending a GET request with an "Observe" option.

REQ: GET example.com/mg/ietf-ip/ipv6/neighbor/ip
(observe option register)

RES: 2.05 Content (Content-Format: application/cbor)
{
  "ip" : "fe80::200:f8ff:fe21:67cf"
}

The same example with the hash values instead of the string identifiers looks like:

REQ: GET example.com/mg/guJB_
(observe option register)

RES: 2.05 Content (Content-Format: application/cbor)
{
  0x20b8907e : "fe80::200:f8ff:fe21:67cf"
}
In the example, the request returns a success response with the contents of the ip field. Consecutively the server will regularly notify the client when ip changes value.

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. Module 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"; obs

In the example the "obs" attribute indicates that the object /mg/ CF_fA is observed.

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 following the RESTCONF protocol. The YANG module information is 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.

Since many constrained servers within a deployment are likely to be similar, the module list can be stored locally on each server, or remotely on a different server.

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.

4.5. 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: complete RESTCONF to CoAP error code mappings
<table>
<thead>
<tr>
<th>RESTCONF Status Line</th>
<th>CoAP Status Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 Continue</td>
<td>none?</td>
</tr>
<tr>
<td>200 OK</td>
<td>2.05</td>
</tr>
<tr>
<td>201 Created</td>
<td>2.01</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>None?</td>
</tr>
<tr>
<td>204 No Content</td>
<td>?</td>
</tr>
<tr>
<td>304 Not Modified</td>
<td>2.03</td>
</tr>
<tr>
<td>400 Bad Request</td>
<td>4.00</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>4.03</td>
</tr>
<tr>
<td>404 Not Found</td>
<td>4.04</td>
</tr>
<tr>
<td>405 Method Not Allowed</td>
<td>4.05</td>
</tr>
<tr>
<td>409 Conflict</td>
<td>None?</td>
</tr>
<tr>
<td>412 Precondition Failed</td>
<td>4.12</td>
</tr>
<tr>
<td>413 Request Entity Too Large</td>
<td>4.13</td>
</tr>
<tr>
<td>414 Request-URI Too Large</td>
<td>4.00</td>
</tr>
<tr>
<td>415 Unsupported Media Type</td>
<td>4.15</td>
</tr>
<tr>
<td>500 Internal Server Error</td>
<td>5.00</td>
</tr>
<tr>
<td>501 Not Implemented</td>
<td>5.01</td>
</tr>
<tr>
<td>503 Service Unavailable</td>
<td>5.03</td>
</tr>
</tbody>
</table>

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 http://en.wikipedia.org/wiki/MurmurHash. Implementation are available online, including at https://code.google.com/p/smhasher/wiki/MurmurHash. 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.

Prefix values are used on every node.

The prefix values defined in the YANG module containing the data object are used for the path expression. For external modules, this is the value of the 'prefix' sub-statement in the 'import' statement for each external module.

Path expressions for objects which augment data nodes in external modules are calculated in the augmenting module, using the prefix values in the augmenting module.

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 "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.

5.2. Re-Hash Procedure

A hash collision occurs if two different path identifier strings have the same hash value. If the server has over 38,000 objects in its YANG modules, then the probability of a collision is fairly high. If a hash collision occurs on the server, then the object 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.
In most cases, the hash function is expected to produce unique values for all the objects 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-map" entry for the altered identifier, as described in the following YANG module.

5.3. 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 ietf-yang-hash
    +--ro rehash* [hash]
        +--ro hash     uint32
        +--ro path?    string
        +--ro append?  string
```

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

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

Copyright (c) 2014 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-05.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2014-10-27 {
  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";
    }

    leaf path {
      type string;
      description
        "The YANG identifier path expression that caused the
        collision and is being remapped";
    }

    leaf append {
      type string;
      description
        "The string that the server appended to the path
        expression contained in the 'path' leaf to produce
        a new path expression and therefore new hash value.
        The YANG hash value for the new string (identified
        by 'path' + 'append') is used to identify the
        'path' object."
    }
  }
}

5.3.1. YANG Re-Hash Example

In this example the server has an object that is already registered
when the "/foo:A/foo:B/foo:col1" object is processed. This object
path string hashes to value 0x29abdcca. The server has appended the
string "\" to the path to produce a new hash (0x2a7a2044) which does
not collide with any other objects.

The server would return the following information if the client
retrieved the "/mg/yang-hash" resource.
REQ: GET example.com/mg/yang-hash

RES: 2.05 Content (Content-Format: application/cbor)
{ "ietf-yang-hash:yang-hash" : {
  "rehash" : [
    {
      "hash" : 712646724,
      "path" :"/foo:A/foo:B/foo:col1",
      "append" : "_"
    }
  ]
}}

5.4. 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. 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.

<table>
<thead>
<tr>
<th>YANG type</th>
<th>CBOR type</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8, int16, int32, int64, uint16, uint32, uint64, decimal64</td>
<td>unsigned int (major type 0) or negative int (major type 1)</td>
<td>The CBOR integer type depends on the sign of the actual value.</td>
</tr>
<tr>
<td>boolean</td>
<td>either &quot;true&quot; (major type 7, simple value 21) or &quot;false&quot; (major type 7, simple value 20)</td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>text string (major type 3)</td>
<td></td>
</tr>
<tr>
<td>enumeration</td>
<td>unsigned int (major type 0)</td>
<td></td>
</tr>
<tr>
<td>bits</td>
<td>array of text strings</td>
<td>Each text string contains the name of a bit value that is set.</td>
</tr>
<tr>
<td>binary</td>
<td>byte string (major type 2)</td>
<td></td>
</tr>
<tr>
<td>empty</td>
<td>null (major type 7, simple value 22)</td>
<td>TBD: This MAY not be applicable to true MIBs, as SNMP may not support empty variables...</td>
</tr>
<tr>
<td>union</td>
<td>Similar ot the JSON transcription from</td>
<td></td>
</tr>
</tbody>
</table>
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              | array (major type 4) of maps (major type 5) | 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.

<table>
<thead>
<tr>
<th>CoMI Error Code</th>
<th>CoAP Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.00</td>
<td>General error</td>
</tr>
<tr>
<td>1</td>
<td>4.00</td>
<td>Malformed CBOR data</td>
</tr>
<tr>
<td>2</td>
<td>4.00</td>
<td>Incorrect CBOR datatype</td>
</tr>
<tr>
<td>3</td>
<td>4.00</td>
<td>Unknown MIB variable</td>
</tr>
<tr>
<td>4</td>
<td>4.00</td>
<td>Unknown conversion table</td>
</tr>
<tr>
<td>5</td>
<td>4.05</td>
<td>Attempt to write read-only variable</td>
</tr>
<tr>
<td>0..2</td>
<td>5.01</td>
<td>Access exceptions</td>
</tr>
<tr>
<td>0..18</td>
<td>5.00</td>
<td>SMI error status</td>
</tr>
</tbody>
</table>

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 authorised 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 for protected access to resources, as well suitable authentication and authorisation 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 authorisation 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.yang-hash"' needs registration with IANA.

Content types to be registered:

- application/comi+cbor

10. Acknowledgements

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 Dee Denteneer, Esko Dijk, Michael van Hartskamp, Zach Shelby, Michel Veillette, Michael Verschoor, and Thomas Watteyne. The CBOR encoding borrows extensively from Ladislav Lhotka’s description on conversion from YANG to JSON.

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

12. References
12.1. Normative References


12.2. Informative References


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:


<table>
<thead>
<tr>
<th>Code</th>
<th>processor</th>
<th>Text</th>
<th>Data</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observe agent</td>
<td>erbium</td>
<td>800</td>
<td>n/a</td>
<td>[Erbium]</td>
</tr>
<tr>
<td>CoAP server</td>
<td>MSP430</td>
<td>1K</td>
<td>6</td>
<td>[openwsn]</td>
</tr>
<tr>
<td>SNMP server</td>
<td>ATmega128</td>
<td>9K</td>
<td>700</td>
<td>[management]</td>
</tr>
<tr>
<td>Secure SNMP</td>
<td>ATmega128</td>
<td>30K</td>
<td>1.5K</td>
<td>[management]</td>
</tr>
<tr>
<td>DTLS server</td>
<td>ATmega128</td>
<td>37K</td>
<td>2K</td>
<td>[management]</td>
</tr>
<tr>
<td>NETCONF</td>
<td>ATmega128</td>
<td>23K</td>
<td>627</td>
<td>[management]</td>
</tr>
<tr>
<td>JSON parser</td>
<td>CC2538</td>
<td>4.6K</td>
<td>8</td>
<td>[dcaf]</td>
</tr>
<tr>
<td>CBOR parser</td>
<td>CC2538</td>
<td>1.5K</td>
<td>2.6K</td>
<td>[dcaf]</td>
</tr>
<tr>
<td>DTLS server</td>
<td>ARM7</td>
<td>15K</td>
<td>4</td>
<td>[I-D.ietf-lwig-coap]</td>
</tr>
<tr>
<td>DTLS server</td>
<td>MSP430</td>
<td>15K</td>
<td>4</td>
<td>[DTLS-size]</td>
</tr>
<tr>
<td>Certificate</td>
<td>MSP430</td>
<td>23K</td>
<td></td>
<td>[DTLS-size]</td>
</tr>
<tr>
<td>Crypto</td>
<td>MSP430</td>
<td>2-8K</td>
<td></td>
<td>[DTLS-size]</td>
</tr>
</tbody>
</table>

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.
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. comparison with LWM2M

TODO: Anuj promised text

Authors’ Addresses

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

Bert Greevenbosch
independent
Email: ietf@bertgreevenbosch.nl

Andy Bierman
YumaWorks
Email: andy@yumaworks.com

Juergen Schoenwaelder
Jacobs University
Campus Ring 1
Bremen 28759
Germany
Email: j.schoenwaelder@jacobs-university.de

Anuj Sehgal
Jacobs University
Campus Ring 1
Bremen 28759
Germany
Email: s.anuj@jacobs-university.de
Abstract

Several applications (for example see [I-D.vanderstok-core-comi]) which extend the Constrained Application Protocol (CoAP) need to perform partial resource modifications. The existing CoAP PUT method only allows a complete replacement of a resource. This proposal adds a new CoAP method, PATCH, 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 September 9, 2015.

Copyright Notice

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

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

This specification defines the new Constrained Application Protocol (CoAP) [RFC7252] method, PATCH, which is 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.

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 mentioned in an early design stage of CoAP but was deemed unnecessarily complicated. With the arrival of the Constrained Management Interface (CoMI) protocol, [I-D.vanderstok-core-comi], the need to do partial changes to resources specified with YANG becomes more acute. Applications might wish to make to changes to parts of a YANG data resource, and transferring all data associated with a YANG data resource unnecessarily burdens 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 Method

The PATCH method requests that a set of changes described in the request payload is applied to the resource identified by the Request-URI. 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, resulting in a 2.01 (Created) Response Code. Restrictions to a PATCH 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 but idempotent conformant to CoAP PUT specified in [RFC7252], Section 5.8.3.

PATCH can use confirmable (CON) or Non-confirmable (NON) CoAP requests. It is recommended to use the CON version of the PATCH command.

A PATCH request is idempotent to prevent bad outcomes from collisions between two PATCH 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]).

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. In general, modifications MUST NOT be executed when an error occurs or only a partial execution is possible. The atomicity requirement holds for all directly affected (sub)resources. See "Response
Codes", Section 2.2, for details on status codes and possible error conditions.

If the request passes through a cache and the Request-URI identifies one or more currently cached responses, those responses SHOULD be treated as being stale. A cached PATCH response can only be used to respond to subsequent GET requests; it MUST NOT be used to respond to other methods (in particular, PATCH).

There is no guarantee that a resource can be modified with PATCH. 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 Request-URI.

Clients MUST choose to use PATCH rather than PUT when the request affects (sub)resources of a given resource.

2.1. A Simple PATCH Example

REQ: PATCH
   coap://www.example.com/object/sub1
   payload with changes
RET:
   CoAP 2.04 Changed

This example illustrates use of a hypothetical PATCH on the sub resource /object/sub1 of the existing resource "object". The 2.04 (Changed) response code is conforms with the CoAP PUT method.

2.2. Response Codes

PATCH for CoAP adopts 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].

2.4. Securing PATCH

PATCH is secured following the CoAP recommendations as specified in section 9 of [RFC7252]. When more appropriate security techniques are standardized for CoAP, PATCH can also be secured by those new techniques.
3. Error Handling

A PATCH request may fail under certain known conditions. These situations should be dealt with as expressed below.

Malformed PATCH payload: If a server determines that the payload provided with a PATCH 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 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 X.XX CoAP error. Such a scenario might include situations when:

* the server has insufficient computing resources to complete the request successfully,
* the resource specified in the request becomes invalid by applying the payload,
* modifying a resource leads to a conflicting state.

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 modification: In situations when a server detects possible conflicting modifications and no precondition is defined in the requests, the server can return a X.XX CoAP status.

Conflicting state: If the modification specified by a PATCH request cannot be applied to a resource in its current state, or causes the resource to enter an inconsistent state the server can return the X.XX CoAP status. Such a situation might be encountered when a structural modification is applied to a configuration datastore, but the structures being modified do not exist or lead the device into an inconsistent state if the modifications are made.

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 X.XX CoAP status.

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 protocol. It is meant to inform protocol and application developers about the security limitations of CoAP PATCH 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 are nearly identical to the security considerations for PUT ([RFC7252]). Whatever mechanisms are used for PUT can be used for PATCH as well.

5. IANA Considerations

The entry with name PATCH in the sub-registry, "CoAP Method Codes", is 0.05. The addition will follow the "IETF Review or IESG Approval" procedure as described in [RFC5226].

6. Acknowledgements

This document reflects discussions and remarks from several individuals including (in alphabetical order):
7. Change log

When published as a RFC, this section needs to be removed.

8. References

8.1. Normative References


8.2. Informative References


Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org
Abstract

6LoWPAN 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 6LoWPAN networks with the goal of guiding and stimulating the discussion on Sleepy Nodes support for CoAP in the CoRE WG.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.
Table of Contents

1. Introduction ............................................. 3
   1.1. Problem statement .................................. 3
   1.2. Assumptions ........................................ 4
   1.3. Requirements Language ............................... 4
2. Solution Architecture ...................................... 6
3. Use case scenarios ......................................... 7
4. Initial operations .......................................... 9
   4.1. Proxy Discovery .................................... 10
   4.2. Registration at a Proxy ............................. 10
   4.3. Initialization of Delegated Resource ............... 10
   4.4. Proxy registers at a Discovery Server on behalf of Sleepy 
        Node .............................................. 11
5. Interfaces during operation .................................. 11
   5.1. Discovering Node DISCOVERs Sleepy Node via Discovery 
        Server .......................................... 11
   5.2. Discovering Node DISCOVERs Sleepy Node via Proxy .... 11
   5.3. Sleepy Node REPORTs events directly to Destination Node . 11
   5.4. Sleepy Node REPORTs event to Destination Node(s) via 
        Proxy ............................................. 12
   5.5. Sleepy Node WRITEs changed resource to Proxy ....... 12
   5.6. A Node WRITEs to Sleepy Node via Proxy .......... 12
   5.7. Sleepy Node READs resource updates from Proxy .... 13
   5.8. A Node READs information from Sleepy Node via Proxy ... 13
   5.9. A Sleepy Node READs information from a Server Node ... 14
6. Realization with PubSub server ............................ 14
7. Acknowledgements .......................................... 14
8. IANA Considerations ....................................... 14
9. Security Considerations ................................... 14
10. References ................................................ 15
    10.1. Normative References .............................. 15
    10.2. Informative References ............................ 15
1. Introduction

6LoWPAN 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 6LoWPAN 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 6LoWPAN 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. As such it contributes to understanding how well the other proposals support the operation of the sleepy nodes in a building control context.

The issues that need to be addressed to provide support for Sleepy Nodes in 6LoWPAN networks are summarized in Section 1.1. Section 2 shows the communications patterns involving Sleepy Nodes in 6LoWPAN networks. Section 3 provides a set of use case descriptions that illustrate how these communication patterns can be used in home and building control scenarios. For each of these scenarios, the behaviour of the Sleepy Node is explained in Section 5.

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 an event 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 6LoWPAN 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],
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.

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].

2. Solution Architecture

The solution architecture described in this document makes use of a Proxy Node to which a Sleepy Node delegates part of its communication tasks during its sleeping periods. In particular, the solution is based on the set of functionalities described in [I-D.vial-core-mirror-server] according to which a Proxy Node hosts a ‘delegated’ version of the original CoAP resources of the Sleepy Node. [I-D.vial-core-mirror-server] provides the interface to register, update and remove proxied resources, along with the interface to read and update the proxied resources by both the Sleepy Node and Regular Nodes.

Figure 1 provides an overview of the communication interfaces required to support a Sleepy Node in a 6LoWPAN Network, highlighting the different types and roles of the Nodes (shown as blocks) along with the interactions between them. The interfaces are depicted as arrows. The arrows point from the Node taking the communication initiative to the target Node.

In some implementations, the roles of Proxy and Discovery Server could be implemented by a single node. Furthermore, a single Node could act in a combination of roles (e.g. it may play both the role of discovering node and Configuring Node).
3. Use case scenarios

To describe the application viewpoint of the solution, we introduce some example scenarios for the various interface functions in Figure 1, assuming the Sleepy Node to be a sensor device in a home or a building control context.

Function 1: a Node DISCOVERs Sleepy Node(s) (via Proxy or Discovery Server); for example:

- A Node wants to discover given services related to a group of deployed sensors via multicast. It gets responses for the sleeping sensors from the Proxy nodes.
- During commissioning phase, a configuring node queries a Discovery Server to find all the proxies providing a given service.

Function 2: Sleepy Node REPORTs event to other Node(s) (directly or via Proxy); for example:
- A battery-powered sensor sends an event "battery low" directly to a designated reporting location Node.

- A battery-powered occupancy sensor detects an event "people present", switches on the radio and sends a request to one or a group of lights to turn on.

- A battery-powered temperature sensor reports periodically the room temperature to a designated Node that controls HVAC devices. The sensor reports also extra events when the temperature change deviates from a predefined range.

Function 3: Sleepy Node WRITEs information to the Proxy; for example:

- A battery-powered sensor wants to extend the registration lifetime of its delegated resource at the Proxy.

Function 4: Sleepy Node READs from other Node(s) (directly or via Proxy); 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.

- 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).

Function 5: Node READs information from Sleepy Node(s) (via Proxy only); 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.

- An authorized 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.
Function 6: A Node WRITEs information to a Sleepy Node (via Proxy only); for example:

- An authorized Node changes the reporting frequency of a deployed sensor by contacting the Proxy node to which the sensor is registered.
- Sensor firmware is upgraded. An authorized Node pushes firmware data blocks to the Proxy, which pushes the blocks to the Sleepy Node.

4. Initial operations

In order to become fully operational in a network and to communicate over the interfaces shown in Figure 1, a Sleepy Node needs first to perform some initial operations:

- Discovery of Proxy (directly or via Discovery Server)
- Registration of resources to delegate at a Proxy
- Initialization of its delegated resources at the Proxy
- Registration to a Discovery Server via Proxy (optional)
4.1. Proxy Discovery

A Sleepy Node can find a Proxy implementing resource cache functionalities to which it can delegate its own resources by means of:

1. Discovery via Discovery Server: this interface is the default one supplied by the Discovery Server, e.g. CoRE Resource Directory [I-D.ietf-core-resource-directory] or DNS-SD [RFC6763].

2. Direct Discovery: a CoAP multicast GET request can be performed on the /.well-known/core resource as specified for CoAP in [RFC7390].

In both cases, a query can be done for the core.ms resource type, defined in [I-D.vial-core-mirror-server].

In a system, The Proxy discovery can be performed even in both ways (e.g. if Discovery via Discovery Server fails, the Sleepy Node can try Direct Discovery).

4.2. Registration at a Proxy

Once a Sleepy Node has discovered a Proxy by means of one of the procedures described above, 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. The description of the resource includes the Sleepy Node identifier, its domain and the lifetime of the registration. The Link Format description is identical to the /.well-known/core resource. At the moment of the registration at the Proxy, the Sleepy Node may specify the ‘obs’ attribute to indicate to the Proxy that a CoAP observation relationship between the delegated resource and a client is allowed and can be performed as described in IETF Draft CoRE Observe [I-D.ietf-core-observe]. Upon successful registration, the Proxy creates a new resource and returns its location.

4.3. Initialization of Delegated Resource

Once registration has been successfully performed, the Sleepy Node must initialize the delegated resource before it can be visible in Resource Discovery via the Proxy Node. 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 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 dele
gated resources by updating its ./well-known/core resource.

4.4. Proxy registers at a Discovery Server on behalf of Sleepy Node

Once a Sleepy Node has registered itself to a Proxy, the Proxy has the responsibility to register the Sleepy Node to a Discovery Server and to keep this registration up-to-date. This interface, not to be confused with the interface in which the Sleepy Node registers its resources to a Proxy, is required whenever a Discovery Server is present in the network. There may be in fact deployments that do not have a Discovery Server. At run-time, the Proxy will try to find a Discovery Server and if such server is found it will register the Sleepy Node. The details of the interface are exactly according to the respective Discovery Server specification. A special case might be when Proxy and Discovery Server are embodied by the same node. In this case the registration occurs as an internal process within the Proxy Node itself, upon registration of the Sleepy Node at the Proxy.

5. Interfaces during operation

This section details the scope and behaviour of each interface function specified in the architecture in Figure 1.

5.1. Discovering Node DISCOVERs Sleepy Node via Discovery Server

Through this interface, a Discovering Node can discover one or more Sleepy Node(s) through a Discovery Server. The interface is the default one supplied by the Discovery Server, e.g. CoRE Resource Directory or DNS-SD.

5.2. Discovering Node DISCOVERs Sleepy Node via Proxy

Through this interface, a Discovering Node can discover one or more Sleepy Node(s) through a Proxy. In case a Discovery Server is not active in a system, 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].

5.3. 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.
5.4. Sleepy Node REPORTs event to Destination Node(s) via Proxy

This interface can be used by the Sleepy Node to communicate a sensor event report message to Proxy (REPORT A) which will further notify it to interested Destination Node(s) (REPORT B) that are not directly present in the subscribers list of the Sleepy Node itself. 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. The standard CoAP unicast POST can be used to report events to the Proxy (REPORT A), while the mechanism according to which the Proxy forwards the event to Destination Nodes (REPORT B) may be linked to a specific protocol (for example: CoAP, HTTP, or publish/subscribe as in MQTT). 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 event to its Proxy, it can perform a subscription at the Proxy to that specific resource. In this case, a standard CoAP GET with the CoAP Observe option on the delegated resource at the Proxy can be used, as described in [I-D.ietf-core-observe].

5.5. Sleepy Node WRITEs changed resource to Proxy

A Sleepy Node can update a proxy resource at the Proxy using a standard CoAP PUT requests on the proxied resource. This interface is only needed when a resource can be changed on the Sleepy Node outside the knowledge of the Proxy, i.e. by an entity which is not the Proxy. For example, a resource can be changed by the Sleepy Node itself. It is good practice, to avoid write/write conflicts at the proxy side, to ensure that such frequently-updated resources are read-only, e.g. the sensed temperature value of a sensor can be read by external nodes but not written.

5.6. 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 PS Node checks the Proxy for any modification of its delegated resources and reads those changed resources using CoAP GET requests, as shown in Figure 3. 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.
5.7. Sleepy Node READs resource updates from Proxy

This interface allows a Sleepy Node to retrieve a list of delegated resources that have been modified at the Proxy by other nodes. As in [I-D.vial-core-mirror-server], the path /ms is used to store the sleepy node resources in the proxy.

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 3 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.

```
+--------+              +-------+               +---------+
| Sleepy |              | Proxy |               | Regular  |
|  Node  |              |       |               |   Node   |
+--------+              +-------+               +---------+
                                 <---PUT /ms/0/dev/n----|
                                 Payload: Sensor1
                                 ---2.04 Changed-------->
                                 Wake-up
                                 event
---POST /ms/0?chk------>    
<----2.04 Changed------>
Payload: <ms/0/dev/n>
---GET /ms/0/dev/n---->     
<------2.05 Content------>
Payload: Sensor1
```

Figure 3: Example: A Sleepy Node READs resource updates from his Proxy

5.8. 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 on 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 5.6) 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.
5.9. 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.

6. Realization with PubSub server

The registration and discovery of the PubSub broker [I-D.koster-core-coap-pubsub] is covered to the same extent as discussed in this document. Not covered is the direct interaction between sleepy node and destination nodes. The support from a server node to initialize resources or other information also represents an addition to PubSub broker.

In addition to the continuous updates provided by the PubSub broker, the ad-hoc query of values, the maintenance of operational parameters, the provision of direct update from sleepy node to a node, the reliability aspects of the update, and the concept of groups are equally important topics that need consideration.

7. Acknowledgements

TBD

8. IANA Considerations

The new Resource Type (rt=) Link Target Attribute, ’core.ms’ needs to be registered in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry. This is not yet done by [I-D.vial-core-mirror-server].

9. Security Considerations

Layer 2 (MAC) security is used in all communication in the 6LoWPAN network. A Sleepy Node may obtain the Layer 2 network key using the bootstrapping mechanism described in [I-D.kumar-6lo-selective-bootstrap]. On top of this, DTLS and DTLS-multicast can be used for further transport-layer protection of messages between a Sleepy Node and other nodes; and also between a Proxy and other nodes. 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).

10. References

10.1. Normative References


10.2. Informative References


Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe in the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-00 (work in progress), October 2014.

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.


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
Kamperfoelie 8
Helmond 5708 DM
The Netherlands

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