Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2016

T. Fossati
Alcatel-Lucent
H. Tschofenig
ARM Ltd.

July 6, 2015

Server Endpoint Identifiers for Certificate Mode (D)TLS
draft-fossati-core-certmode-rd-names-01

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

Copyright Notice

Table of Contents

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:

- o the dot ("."), which is used to introduce the domain component (REQ#3);

- o 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 then 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

- o eui-64+01-23-45-67-89-ab-cd-ef

- o imei+123456789012345

- o imei+123456789012345:9876

- o uuid+64d5ecfa-addc-4695-ac6e-36e8b18de4b9

- o eui-64+01-23-45-67-89-ab-cd-ef.local:1234

- o 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:

```
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:

```
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':

```
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)?]]

- o id-coap
- o OtherName.type-id::id-rdauthority-san
- o NameType::rd_authority
- o 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

[EUI-64] IEEE, "Guidelines for 64-bit Global Identifier (EUI-64)", November 2012.

[I-D.ietf-core-resource-directory] Shelby, Z. and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-02 (work in progress), November 2014.

[I-D.ietf-dice-profile] Tschofenig, H., "A Datagram Transport Layer Security (DTLS) 1.2 Profile for the Internet of Things", draft-ietf-dice-profile-04 (work in progress), August 2014.

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

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

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

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

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 Security (TLS)", RFC 6125, March 2011.

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

C. Bormann
Universitaet Bremen TZI
Z. Shelby, Ed.
ARM

October 27, 2014

Blockwise transfers in CoAP
draft-ietf-core-block-16

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

Copyright Notice

Table of Contents

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:

- o Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.

- o No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.

- o The transfer of each block is acknowledged, enabling individual retransmission if required.

- o Both sides have a say in the block size that actually will be used.

- o The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.

- o If needed, the Block options can also be used (without changes) to provide random access to power-of-two sized blocks within a resource representation.

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

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

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

2. Block-wise transfers

As discussed in the introduction, there are good reasons to limit the size of datagrams in constrained networks:

- o by the maximum datagram size (~ 64 KiB for UDP)
- o by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- o by the desire to avoid adaptation layer fragmentation (60-80 bytes for 6LoWPAN [RFC4919])

When a resource representation is larger than can be comfortably transferred in the payload of a single CoAP datagram, a Block option can be used to indicate a block-wise transfer. As payloads can be sent both with requests and with responses, this specification provides two separate options for each direction of payload transfer. In 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

No.	C	U	N	R	Name	Format	Length	Default
23	C	U	-	-	Block2	uint	0-3	(none)
27	C	U	-	-	Block1	uint	0-3	(none)

Table 1: Block Option Numbers

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

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

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

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

2.2. Structure of a Block Option

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

- o The size of the block (SZX);
- o whether more blocks are following (M);
- o the relative number of the block (NUM) within a sequence of blocks with the given size.

The value of the Block Option is a variable-size (0 to 3 byte) unsigned integer (uint, see Section 3.2 of [RFC7252]). This integer

value encodes these three fields, see Figure 1. (Due to the CoAP uint encoding rules, when all of NUM, M, and SZX happen to be zero, a zero-byte integer will be sent.)

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

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

 0                   1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   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 sent as zero and ignored on reception. (In a Block1 Option in a response, the M flag is used to indicate atomicity, see below.)

SZX: Block Size. The block size is represented as three-bit unsigned integer indicating the size of a block to the power of two. Thus block size = 2**(SZX + 4). The allowed values of SZX are 0 to 6, i.e., the minimum block size is 2**(0+4) = 16 and the maximum is 2**(6+4) = 1024. The value 7 for SZX (which would indicate a block size of 2048) is reserved, i.e. MUST NOT be sent and MUST lead to a 4.00 Bad Request response code upon reception in a request.

There is no default value for the Block1 and Block2 Options. Absence of one of these options is equivalent to an option value of 0 with respect to the value of NUM and M that could be given in the option, i.e. it indicates that the current block is the first and only block of the transfer (block number 0, M bit not set). However, in contrast to the explicit value 0, which would indicate an SZX of 0 and thus a size value of 16 bytes, there is no specific explicit size implied by the absence of the option -- the size is left unspecified. (As for any uint, the explicit value 0 is efficiently indicated by a zero-length option; this, therefore, is different in semantics from the absence of the option.)

2.3. Block Options in Requests and Responses

The Block options are used in one of three roles:

- o In descriptive usage, i.e., a Block2 Option in a response (such as a 2.05 response for GET), or a Block1 Option in a request (a PUT or POST):
  - * The NUM field in the option value describes what block number is contained in the payload of this message.
  - * The M bit indicates whether further blocks need to be transferred to complete the transfer of that body.

* The block size implied by SZX MUST match the size of the payload in bytes, if the M bit is set. (SZX does not govern the payload size if M is unset). For Block2, if the request suggested a larger value of SZX, the next request MUST move SZX down to the size given in the response. (The effect is that, if the server uses the smaller of (1) its preferred block size and (2) the block size requested, all blocks for a body use the same block size.)

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

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

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

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

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

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

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

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

  + Conversely, if the M bit is unset even though it was set in the request, it indicates the block-wise request was enacted now specifically for this block, and the response carries the final response to this request (and to any previous ones with the M bit set in the response's Block1 Option in this sequence of block-wise transfers); the client is still

expected to continue sending further blocks, the request method for which may or may not also be enacted per-block.

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

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

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

2.4. Using the Block2 Option

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

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

Once the Block2 Option is used by the requester and a first response has been received with a possibly adjusted block size, all further requests in a single block-wise transfer SHOULD ultimately use the same size, except that there may not be enough content to fill the

last block (the one returned with the M bit not set). (Note that the client may start using the Block2 Option in a second request after a first request without a Block2 Option resulted in a Block2 option in the response.) The server SHOULD use the block size indicated in the request option or a smaller size, but the requester MUST take note of the actual block size used in the response it receives to its initial request and proceed to use it in subsequent requests. The server behavior MUST ensure that this client behavior results in the same block size for all responses in a sequence (except for the last one with the M bit not set, and possibly the first one if the initial request did not contain a Block2 Option).

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

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

2.5. Using the Block1 Option

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

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

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

In a 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.

```
CLIENT                                                   SERVER
  |                                                        |
  | CON [MID=1234], GET, /status                      ------> |
  |                                                        |
  | <------   ACK [MID=1234], 2.05 Content, 2:0/1/128      |
  |                                                        |
  | CON [MID=1235], GET, /status, 2:1/0/128           ------> |
  |                                                        |
  | <------   ACK [MID=1235], 2.05 Content, 2:1/1/128      |
  |                                                        |
  | CON [MID=1236], GET, /status, 2:2/0/128           ------> |
  |                                                        |
  | <------   ACK [MID=1236], 2.05 Content, 2:2/0/128      |
```

Figure 2: Simple blockwise GET

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                                                   SERVER
  |                                                        |
  | 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=1238], GET, /status, 2:4/0/64            ------> |
  |                                                        |
  | <------   ACK [MID=1238], 2.05 Content, 2:4/1/64       |
  |                                                        |
  | CON [MID=1239], GET, /status, 2:5/0/64            ------> |
  |                                                        |
  | <------   ACK [MID=1239], 2.05 Content, 2:5/0/64       |
```

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.

```
CLIENT                                                   SERVER
  |                                                        |
  | CON [MID=1234], GET, /status                     ------> |
  |                                                        |
  | <------   ACK [MID=1234], 2.05 Content, 2:0/1/128      |
  |                                                        |
  | CON [MID=1235], GET, /status, 2:2/0/64           ------> |
  |                                                        |
  | <------   ACK [MID=1235], 2.05 Content, 2:2/1/64       |
  |                                                        |
  | CON [MID=1236], GET, /status, 2:3/0/64           ------> |
  |                                                        |
  | <------   ACK [MID=1236], 2.05 Content, 2:3/1/64       |
  |                                                        |
  | CON [MID=1237], GET, /status, 2:4/0/64           ------> |
  |                                                        |
  | <------   ACK [MID=1237], 2.05 Content, 2:4/1/64       |
  |                                                        |
  | CON [MID=1238], GET, /status, 2:5/0/64           ------> |
  |                                                        |
  | <------   ACK [MID=1238], 2.05 Content, 2:5/0/64       |
```

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).

```
   CLIENT                                                   SERVER
     |                                                        |
     | CON [MID=1234], GET, /status                     ------> |
     |                                                        |
     | <------   ACK [MID=1234], 2.05 Content, 2:0/1/128      |
     |                                                        |
     | CON [MID=1235], GE//////////////////////               |
     |                                                        |
     | (timeout)                                              |
     |                                                        |
     | CON [MID=1235], GET, /status, 2:2/0/64           ------> |
     |                                                        |
     | <------   ACK [MID=1235], 2.05 Content, 2:2/1/64       |
     :                                                        :
     :                          ...                           :
     :                                                        :
     | CON [MID=1238], GET, /status, 2:5/0/64           ------> |
     |                                                        |
     | <------   ACK [MID=1238], 2.05 Content, 2:5/0/64       |
```

Figure 5: Blockwise GET with late negotiation and lost CON

```
   CLIENT                                                   SERVER
     |                                                        |
     | CON [MID=1234], GET, /status                     ------> |
     |                                                        |
     | <------   ACK [MID=1234], 2.05 Content, 2:0/1/128      |
     |                                                        |
     | CON [MID=1235], GET, /status, 2:2/0/64           ------> |
     |                                                        |
     | ////////////////////////////////tent, 2:2/1/64          |
     |                                                        |
     | (timeout)                                              |
     |                                                        |
     | CON [MID=1235], GET, /status, 2:2/0/64           ------> |
     |                                                        |
     | <------   ACK [MID=1235], 2.05 Content, 2:2/1/64       |
     :                                                        :
     :                          ...                           :
     :                                                        :
     | CON [MID=1238], GET, /status, 2:5/0/64           ------> |
     |                                                        |
     | <------   ACK [MID=1238], 2.05 Content, 2:5/0/64       |
```

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.

```
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 blockwise PUT

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.

```
CLIENT                                                  SERVER
  |                                                       |
  | CON [MID=1234], PUT, /options, 1:0/1/128    ------>   |
  |                                                       |
  | <------   ACK [MID=1234], 2.04 Changed, 1:0/1/32      |
  |                                                       |
  | CON [MID=1235], PUT, /options, 1:4/1/32     ------>   |
  |                                                       |
  | <------   ACK [MID=1235], 2.04 Changed, 1:4/1/32      |
  |                                                       |
  | CON [MID=1236], PUT, /options, 1:5/1/32     ------>   |
  |                                                       |
  | <------   ACK [MID=1235], 2.04 Changed, 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 blockwise PUT with negotiation

3.3. Combining Block1 and Block2

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

```
CLIENT                                                    SERVER
  |                                                          |
  | CON [MID=1234], POST, /soap, 1:0/1/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1234], 2.31 Continue, 1:0/1/128       |
  |                                                          |
  | CON [MID=1235], POST, /soap, 1:1/1/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1235], 2.31 Continue, 1:1/1/128       |
  |                                                          |
  | CON [MID=1236], POST, /soap, 1:2/0/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1236], 2.04 Changed, 2:0/1/128, 1:2/0/128 |
  |                                                          |
  | CON [MID=1237], POST, /soap, 2:1/0/128      ------>      |
  | (no payload for requests with Block2 with NUM != 0)      |
  | (could also do late negotiation by requesting e.g. 2:2/0/64) |
  |                                                          |
  | <------   ACK [MID=1237], 2.04 Changed, 2:1/1/128        |
  |                                                          |
  | CON [MID=1238], POST, /soap, 2:2/0/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1238], 2.04 Changed, 2:2/1/128        |
  |                                                          |
  | CON [MID=1239], POST, /soap, 2:3/0/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1239], 2.04 Changed, 2:3/0/128        |
```

Figure 10: Atomic blockwise POST with blockwise response

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

```
   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 blockwise POST with blockwise response, early negotiation

3.4. Combining Observe and Block2

In the following example, the server first sends a direct response (Observe sequence number 62350) to the initial GET request (the resulting 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.

```
   CLIENT  SERVER
     |      |
     +----->|     Header: GET 0x41011636
     | GET  |      Token: 0xfb
     |      |   Uri-Path: status-icon
     |      |    Observe: (empty)
     |      |
     |<-----+     Header: 2.05 0x61451636
     | 2.05 |      Token: 0xfb
```

```
          |       |     Block2: 0/1/128
          |       |    Observe: 62350
          |       |       ETag: 6f00f38e
          |       |    Payload: [128 bytes]
          |       |
          |       |  (Usual GET transfer left out)
            ...
          |       |  (Notification of first block:)
          |       |
          <-----+     Header: 2.05 0x4145af9c
           2.05 |      Token: 0xfb
          |       |     Block2: 0/1/128
          |       |    Observe: 62354
          |       |       ETag: 6f00f392
          |       |    Payload: [128 bytes]
          |       |
          +- - ->|     Header: 0x6000af9c
          |       |
          |       |  (Retrieval of remaining blocks)
          |       |
          +----->|     Header: GET 0x41011637
           GET    |      Token: 0xfc
          |       |   Uri-Path: status-icon
          |       |     Block2: 1/0/128
          |       |
          <-----+     Header: 2.05 0x61451637
           2.05 |      Token: 0xfc
          |       |     Block2: 1/1/128
          |       |       ETag: 6f00f392
          |       |    Payload: [128 bytes]
          |       |
          +----->|     Header: GET 0x41011638
           GET    |      Token: 0xfc
          |       |   Uri-Path: status-icon
          |       |     Block2: 2/0/128
          |       |
          <-----+     Header: 2.05 0x61451638
           2.05 |      Token: 0xfc
          |       |     Block2: 2/0/128
          |       |       ETag: 6f00f392
          |       |    Payload: [53 bytes]
```

Figure 12: Observe sequence with 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
      | GET   |      Token: 0xfb
      |       |   Uri-Path: status-icon
      |       |    Observe: (empty)
      |       |     Block2: 0/0/64
      |       |
      |<-----+     Header: 2.05 0x61451636
      | 2.05  |      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:)
      |       |
      |<-----+     Header: 2.05 0x4145af9c
      | 2.05  |      Token: 0xfb
      |       |     Block2: 0/1/64
      |       |    Observe: 62354
      |       |       ETag: 6f00f392
      |       |    Payload: [64 bytes]
      |       |
      +- - ->|     Header: 0x6000af9c
      |       |
      |       |  (Retrieval of remaining blocks)
      |       |
      +----->|     Header: GET 0x41011637
      | GET   |      Token: 0xfc
      |       |   Uri-Path: status-icon
      |       |     Block2: 1/0/64
      |       |
      |<-----+     Header: 2.05 0x61451637
      | 2.05  |      Token: 0xfc
      |       |     Block2: 1/1/64
      |       |       ETag: 6f00f392
```

```
         |      |    Payload: [64 bytes]
           ....
         |      |
         +----->|     Header: GET 0x41011638
         | GET  |      Token: 0xfc
         |      |   Uri-Path: status-icon
         |      |     Block2: 4/0/64
         |      |
         |<-----+     Header: 2.05 0x61451638
         | 2.05 |      Token: 0xfc
         |      |     Block2: 4/0/64
         |      |       ETag: 6f00f392
         |      |    Payload: [53 bytes]
```

Figure 13: Observe sequence with early negotiation

4. The Size2 and Size1 Options

In many cases when transferring a large resource representation block by block, it is advantageous to know the total size early in the process. Some indication may be available from the maximum size estimate attribute "sz" provided in a resource description [RFC6690]. However, the size may vary dynamically, so a more up-to-date indication may be useful.

This specification defines two CoAP Options, Size1 for indicating the size of the representation transferred in requests, and Size2 for indicating the size of the representation transferred in responses. (Size1 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").

- o in a 4.13 response, to indicate the maximum size that would have been acceptable [RFC7252], measured in bytes.

Apart from conveying/asking for size information, the Size options have no other effect on the processing of the request or response. If the client wants to minimize the size of the payload in the resulting response, it should add a Block2 option to the request with a small block size (e.g., setting SZX=0).

The Size Options are "elective", i.e., a client MUST be prepared for the server to ignore the size estimate request. The Size Options MUST NOT occur more than once.

No.	C	U	N	R	Name	Format	Length	Default
60			x		Size1	uint	0-4	(none)
28			x		Size2	uint	0-4	(none)

Table 2: Size Option Numbers

Implementation Notes:

- o As a quality of implementation consideration, 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.

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

- o As usual for an option of type uint, the value 0 is best expressed as an empty option (0 bytes). There is no default value for either Size Option.

- o The Size Options are neither critical nor unsafe, and are marked as No-Cache-Key.

5. HTTP Mapping Considerations

In this subsection, we give some brief examples for the influence the Block options might have on intermediaries that map between CoAP and HTTP.

For mapping CoAP requests to HTTP, the intermediary may want to map the sequence of block-wise transfers into a single HTTP transfer. E.g., for a GET request, the intermediary could perform the HTTP request once the first block has been requested and could then fulfill all further block requests out of its cache. A constrained implementation may not be able to cache the entire object and may use a combination of TCP flow control and (in particular if timeouts occur) HTTP range requests to obtain the information necessary for the next block transfer at the right time.

For PUT or POST requests, historically there was more variation in how HTTP servers might implement ranges; recently, [RFC7233] has defined that Range header fields received with a request method other than GET are not to be interpreted. So, in general, the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer for these other methods within one HTTP request. If enough buffering is available, this request can be started when the last CoAP block is received. A constrained implementation may want to relieve its buffering by already starting to send the HTTP request at the time the first CoAP block is received; any HTTP 408 status code that indicates that the HTTP server became impatient with the resulting transfer can then be mapped into a CoAP 4.08 response code (similarly, 413 maps to 4.13).

For mapping HTTP to CoAP, the intermediary may want to map a single HTTP transfer into a sequence of block-wise transfers. If the HTTP client is too slow delivering a request body on a PUT or POST, the CoAP server might time out and return a 4.08 response code, which in turn maps well to an HTTP 408 status code (again, 4.13 maps to 413). HTTP range requests received on the HTTP side may be served out of a cache and/or mapped to GET requests that request a sequence of blocks overlapping the range.

(Note that, while the semantics of CoAP 4.08 and HTTP 408 differ, this difference is largely due to the different way the two protocols are mapped to transport. HTTP has an underlying TCP connection, which supplies connection state, so a HTTP 408 status code can immediately be used to indicate that a timeout occurred during transmitting a request through that active TCP connection. The CoAP 4.08 response code indicates one or more missing blocks, which may be due to timeouts or resource constraints; as there is no connection state, there is no way to deliver such a response immediately; instead, it is delivered on the next block transfer. Still, HTTP 408 is probably the best mapping back to HTTP, as the timeout is the most likely cause for a CoAP 4.08. Note that there is no way to distinguish a timeout from a missing block for a server without creating additional state, the need for which we want to avoid.)

6. IANA Considerations

This draft adds the following option numbers to the CoAP Option Numbers registry of [RFC7252]:

Number	Name	Reference
23	Block2	[RFCXXXX]
27	Block1	[RFCXXXX]
28	Size2	[RFCXXXX]

Table 3: CoAP Option Numbers

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

Code	Description	Reference
2.31	Continue	[RFCXXXX]
4.08	Request Entity Incomplete	[RFCXXXX]

Table 4: CoAP Response Codes

7. Security Considerations

Providing access to blocks within a resource may lead to surprising vulnerabilities. Where requests are not implemented atomically, an attacker may be able to exploit a race condition or confuse a server by inducing it to use a partially updated resource representation. Partial transfers may also make certain problematic data invisible to intrusion detection systems; it is RECOMMENDED that an intrusion detection system (IDS) that analyzes resource representations transferred by CoAP implement the Block options to gain access to entire resource representations. Still, approaches such as transferring even-numbered blocks on one path and odd-numbered blocks on another path, or even transferring blocks multiple times with different content and obtaining a different interpretation of temporal order at the IDS than at the server, may prevent an IDS from seeing the whole picture. These kinds of attacks are well understood from IP fragmentation and TCP segmentation; CoAP does not add fundamentally new considerations.

Where access to a resource is only granted to clients making use of specific security associations, all blocks of that resource MUST be subject to the same security checks; it MUST NOT be possible for unprotected exchanges to influence blocks of an otherwise protected resource. As a related consideration, where object security is employed, PUT/POST should be implemented in the atomic fashion, unless the object security operation is performed on each access and the creation of unusable resources can be tolerated.

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]
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 Requirement Levels", BCP 14, RFC 2119, March 1997.

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

9.2. Informative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

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

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

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

[RFC7233] Fielding, R., Lafon, Y., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, June 2014.

Authors' Addresses

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

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

Zach Shelby (editor)
ARM
150 Rose Orchard
San Jose, CA 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: January 9, 2017

C. Bormann
Universitaet Bremen TZI
Z. Shelby, Ed.
ARM

July 08, 2016

Block-wise transfers in CoAP
draft-ietf-core-block-21

Abstract

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

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

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

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

A CoAP implementation that does not support these options generally is limited in the size of the representations that can be exchanged. There is therefore an expectation that the Block options are very widely implemented in CoAP implementations, which is why this specification is listed as "updating" RFC 7252.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes (such as microcontrollers with limited RAM and ROM [RFC7228]) and networks (such as 6LoWPAN, [RFC4944]) [RFC7252]. The CoAP protocol is intended to provide RESTful [REST] services not unlike HTTP [RFC7230], while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

This objective requires restraint in a number of sometimes conflicting ways:

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

- o reducing message sizes in order to minimize the number of fragments needed for each message (in turn to maximize the probability of delivery of the message), the amount of transmission power needed and the loading of the limited-bandwidth channel,

- o reducing requirements on the environment such as stable storage, good sources of randomness or user interaction capabilities.

CoAP is based on datagram transports such as UDP, which limit the maximum size of resource representations that can be transferred without creating unreasonable levels of IP fragmentation. In addition, not all resource representations will fit into a single

link layer packet of a constrained network, which may cause adaptation layer fragmentation even if IP layer fragmentation is not required. Using fragmentation (either at the adaptation layer or at the IP layer) for the transport of larger representations would be possible up to the maximum size of the underlying datagram protocol (such as UDP), but the fragmentation/reassembly process burdens the lower layers with conversation state that is better managed in the application layer.

The present specification defines a pair of CoAP options to enable _block-wise_ access to resource representations. The Block options provide a minimal way to transfer larger resource representations in a block-wise fashion. The overriding objective is to avoid the need for creating conversation state at the server for block-wise GET requests. (It is impossible to fully avoid creating conversation state for POST/PUT, if the creation/replacement of resources is to be atomic; where that property is not needed, there is no need to create server conversation state in this case, either.)

Block-wise transfers are realized as combinations of exchanges, each of which is performed according to the CoAP base protocol [RFC7252]. Each exchange in such a combination is governed by the specifications in [RFC7252], including the congestion control specifications (Section 4.7 of [RFC7252]) and the security considerations (Section 11 of [RFC7252]; additional security considerations then apply to the transfers as a whole, see Section 7). The present specification minimizes the constraints it adds to those base exchanges; however, not all variants of using CoAP are very useful inside a block-wise transfer (e.g., using Non-confirmable requests within block-wise transfers outside the use case of Section 2.8 would escalate the overall non-delivery probability). To be perfectly clear, the present specification also does not remove any of the constraints posed by the base specification it is strictly layered on top of; e.g., back-to-back packets are limited by Section 4.7 of [RFC7252] (NSTART as a limit for initiating exchanges, PROBING_RATE as a limit for sending with no response): block-wise transfers cannot send/solicit more traffic than a client could be sending to the same server without the block-wise mode.

In some cases, the present specification will RECOMMEND that a client perform a sequence of block-wise transfers "without undue delay". This cannot be phrased as an interoperability requirement, but is an expectation on implementation quality. Conversely, the expectation is that servers will not have go out of their way to accommodate clients that take forever to finish a block-wise transfer. E.g., for a block-wise GET, if the resource changes while this proceeds, the ETag for a further block obtained may be different. To avoid this happening all the time for a fast-changing resource, a server MAY try

to keep a cache around for a specific client for a short amount of time. The expectation here is that the lifetime for such a cache can be kept short, on the order of a few expected round-trip times, counting from the previous block transferred.

In summary, this specification adds a pair of Block options to CoAP that can be used for block-wise transfers. Benefits of using these options include:

- o Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.

- o No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.

- o The transfer of each block is acknowledged, enabling individual retransmission if required.

- o Both sides have a say in the block size that actually will be used.

- o The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.

- o If needed, the Block options can also be used (without changes) to provide random access to power-of-two sized blocks within a resource representation.

A CoAP implementation that does not support these options generally is limited in the size of the representations that can be exchanged, see Section 4.6 of [RFC7252]. Even though the options are Critical, a server may decide to start using them in an unsolicited way in a response. No effort was expended to provide a capability indication mechanism supporting that decision: since the block-wise transfer mechanisms are so fundamental to the use of CoAP for representations larger than about a kilobyte, there is an expectation that they are very widely implemented.

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

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

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

2. Block-wise transfers

As discussed in the introduction, there are good reasons to limit the size of datagrams in constrained networks:

- o by the maximum datagram size (~ 64 KiB for UDP)
- o by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- o by the desire to avoid adaptation layer fragmentation (60-80 bytes for 6LoWPAN [RFC4919])

When a resource representation is larger than can be comfortably transferred in the payload of a single CoAP datagram, a Block option can be used to indicate a block-wise transfer. As payloads can be sent both with requests and with responses, this specification provides two separate options for each direction of payload transfer. In naming these options (for block-wise transfers as well as in Section 4), we use the number 1 ("Block1", "Size1") to refer to the transfer of the resource representation that pertains to the request, and the number 2 ("Block2", "Size2") to refer to the transfer of the resource representation for the response.

In the following, the term "payload" will be used for the actual content of a single CoAP message, i.e. a single block being transferred, while the term "body" will be used for the entire resource representation that is being transferred in a block-wise fashion. The Content-Format option applies to the body, not to the payload, in particular the boundaries between the blocks may be in places that are not separating whole units in terms of the structure, encoding, or content-coding used by the Content-Format. (Similarly, the ETag option defined in Section 5.10.6 of [RFC7252] applies to the whole representation of the resource and thus to the body of the response.)

In most cases, all blocks being transferred for a body (except for the last one) will be of the same size. (If the first request uses a bigger block size than the receiver prefers, subsequent requests will use the preferred block size.) The block size is not fixed by the protocol. To keep the implementation as simple as possible, the Block options support only a small range of power-of-two block sizes, from 2**4 (16) to 2**10 (1024) bytes. As bodies often will not evenly divide into the power-of-two block size chosen, the size need not be reached in the final block (but even for the final block, the

chosen power-of-two size will still be indicated in the block size field of the Block option).

2.1. The Block2 and Block1 Options

No.	C	U	N	R	Name	Format	Length	Default
23	C	U	-	-	Block2	uint	0-3	(none)
27	C	U	-	-	Block1	uint	0-3	(none)

Table 1: Block Option Numbers

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

Hence, for the methods defined in [RFC7252], Block1 is useful with the payload-bearing POST and PUT requests and their responses. Block2 is useful with GET, POST, and PUT requests and their payload-bearing responses (2.01, 2.02, 2.04, 2.05 -- see Section 5.5 of [RFC7252]).

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

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

2.2. Structure of a Block Option

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

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

o the relative number of the block (NUM) within a sequence of blocks with the given size.

The value of the Block Option is a variable-size (0 to 3 byte) unsigned integer (uint, see Section 3.2 of [RFC7252]). This integer value encodes these three fields, see Figure 1. (Due to the CoAP uint encoding rules, when all of NUM, M, and SZX happen to be zero, a zero-byte integer will be sent.)

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

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

 0                   1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   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 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.)

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

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

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

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

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

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

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

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

  + Conversely, if the M bit is unset even though it was set in the request, it indicates the block-wise request was enacted now specifically for this block, and the response carries the final response to this request (and to any previous ones

with the M bit set in the response's Block1 Option in this sequence of block-wise transfers); the client is still expected to continue sending further blocks, the request method for which may or may not also be enacted per-block. (Note that the resource is now in a partially updated state; this approach is only appropriate where exposing such an intermediate state is acceptable. The client can reduce the window by quickly continuing to update the resource, or, in case of failure, restarting the update.)

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

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

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

2.4. Using the Block2 Option

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

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

size that was used by the server in the response for the first request that gave a desired size using a Block2 Option.

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

Block-wise transfers can be used to GET resources the representations of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing resources. In the latter case, the Block2 Option SHOULD be used in conjunction with the ETag Option ([RFC7252], Section 5.10.6), to ensure that the blocks being reassembled are from the same version of the representation: The server SHOULD include an ETag option in each response. If an ETag option is available, the client, when reassembling the representation from the blocks being exchanged, MUST compare ETag Options. If the ETag Options do not match in a GET transfer, the requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server MAY cache the current value of a representation for an ongoing sequence of requests. (The server may identify the sequence by the combination of the requesting end-point and the URI being the same in each block-wise request.) Note well that this specification makes no requirement for the server to establish any state; however, servers that offer quickly changing resources may thereby make it impossible for a client to ever retrieve a consistent set of blocks. Clients that want to retrieve all blocks of a resource SHOULD strive to do so without undue delay. Servers can fully expect to be free to discard any cached state after a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) after the last access to the state, however, there is no requirement to always keep the state for as long.

The Block2 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise response payload transfer (e.g., GET) operations to the same resource. This is rarely a

requirement, but as a workaround, a client may vary the cache key (e.g., by using one of several URIs accessing resources with the same semantics, or by varying a proxy-safe elective option).

2.5. Using the Block1 Option

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

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

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

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

One reason that a client might encounter a 4.08 error code is that the server has already timed out and discarded the partial request body being assembled. Clients SHOULD strive to send all blocks of a

request without undue delay. Servers can fully expect to be free to discard any partial request body when a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) has elapsed after the most recent block was transferred; however, there is no requirement on a server to always keep the partial request body for as long.

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

A block-wise transfer of a request payload that is implemented in a stateless fashion at the server is likely to leave the resource being operated on in an inconsistent state during the time the transfer is still ongoing or when the client does not complete the transfer. This characteristic is closer to that of remote file systems than to that of HTTP, where state is always kept on the server during a transfer. Techniques well known from shared file access (e.g., client-specific temporary resources) can be used to mitigate this difference from HTTP.

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

2.6. Combining Block-wise Transfers with the Observe Option

The Observe Option provides a way for a client to be notified about changes over time of a resource [RFC7641]. Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. The following rules apply to the combination of block-wise transfers with notifications.

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

As with basic GET transfers, the client can indicate its desired block size in a Block2 Option in the GET request establishing or renewing the observation relationship. If the server supports block-

wise transfers, it SHOULD take note of the block size and apply it as a maximum size to all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the entry in that list is updated by the server receiving a new GET request for the resource from the client).

When sending a 2.05 (Content) notification, the server only sends the first block of the representation. The client retrieves the rest of the representation as if it had caused this first response by a GET request, i.e., by using additional GET requests with Block2 options containing NUM values greater than zero. (This results in the transfer of the entire representation, even if only some of the blocks have changed with respect to a previous notification.)

As with other dynamically changing resources, to ensure that the blocks being reassembled are from the same version of the representation, the server SHOULD include an ETag option in each response, and the reassembling client MUST compare the ETag options (Section 2.4). Even more so than for the general case of Block2, clients that want to retrieve all blocks of a resource they have been notified about with a first block SHOULD strive to do so without undue delay.

See Section 3.4 for examples.

2.7. Combining Block1 and Block2

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

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

2.8. Combining Block2 with Multicast

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

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

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

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

2.9. Response Codes

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

2.9.1. 2.31 Continue

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

2.9.2. 4.08 Request Entity Incomplete

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

(Note that one reason for not having the necessary blocks at hand may be a Content-Format mismatch, see Section 2.3. Implementation note: A server can reject a Block1 transfer with this code when NUM != 0 and a different Content-Format is indicated than expected from the current state of the resource. If it implements the transfer in a stateless fashion, it can match up the Content-Format of the block against that of the existing resource. If it implements the transfer in an atomic fashion, it can match up the block against the partially reassembled piece of representation that is going to replace the state of the resource.)

2.9.3. 4.13 Request Entity Too Large

In [RFC7252], Section 5.9.2.9, the response code 4.13 (Request Entity Too Large) is defined to be like HTTP 413 "Request Entity Too Large". [RFC7252] also recommends that this response SHOULD include a Size1 Option (Section 4) to indicate the maximum size of request entity the

server is able and willing to handle, unless the server is not in a position to make this information available.

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

2.10. Caching Considerations

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

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

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

3. Examples

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

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

As in [RFC7252], "MID" is used as an abbreviation of "Message ID".

3.1. Block2 Examples

The first example (Figure 2) shows a GET request that is split into three blocks. The server proposes a block size of 128, and the client agrees. The first two ACKs contain a payload of 128 bytes each, and the third ACK contains a payload between 1 and 128 bytes.

```
CLIENT                                                    SERVER
  |                                                          |
  | CON [MID=1234], GET, /status                      ------> |
  |                                                          |
  | <------   ACK [MID=1234], 2.05 Content, 2:0/1/128        |
  |                                                          |
  | CON [MID=1235], GET, /status, 2:1/0/128           ------> |
  |                                                          |
  | <------   ACK [MID=1235], 2.05 Content, 2:1/1/128        |
  |                                                          |
  | CON [MID=1236], GET, /status, 2:2/0/128           ------> |
  |                                                          |
  | <------   ACK [MID=1236], 2.05 Content, 2:2/0/128        |
```

Figure 2: Simple block-wise GET

In the second example (Figure 3), the client anticipates the block-wise transfer (e.g., because of a size indication in the link-format description [RFC6690]) and sends a block size proposal. All ACK messages except for the last carry 64 bytes of payload; the last one carries between 1 and 64 bytes.

```
CLIENT                                                   SERVER
  |                                                        |
  | 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=1238], GET, /status, 2:4/0/64          ------> |
  |                                                        |
  | <------   ACK [MID=1238], 2.05 Content, 2:4/1/64        |
  |                                                        |
  | CON [MID=1239], GET, /status, 2:5/0/64          ------> |
  |                                                        |
  | <------   ACK [MID=1239], 2.05 Content, 2:5/0/64        |
```

Figure 3: Block-wise GET with early negotiation

In the third example (Figure 4), the client is surprised by the need for a block-wise transfer, and unhappy with the size chosen unilaterally by the server. As it did not send a size proposal initially, the negotiation only influences the size from the second message exchange onward. Since the client already obtained both the first and second 64-byte block in the first 128-byte exchange, it goes on requesting the third 64-byte block ("2/0/64"). None of this is (or needs to be) understood by the server, which simply responds to the requests as it best can.

```
CLIENT                                                   SERVER
  |                                                        |
  | CON [MID=1234], GET, /status                     ------> |
  |                                                        |
  | <------   ACK [MID=1234], 2.05 Content, 2:0/1/128      |
  |                                                        |
  | CON [MID=1235], GET, /status, 2:2/0/64           ------> |
  |                                                        |
  | <------   ACK [MID=1235], 2.05 Content, 2:2/1/64       |
  |                                                        |
  | CON [MID=1236], GET, /status, 2:3/0/64           ------> |
  |                                                        |
  | <------   ACK [MID=1236], 2.05 Content, 2:3/1/64       |
  |                                                        |
  | CON [MID=1237], GET, /status, 2:4/0/64           ------> |
  |                                                        |
  | <------   ACK [MID=1237], 2.05 Content, 2:4/1/64       |
  |                                                        |
  | CON [MID=1238], GET, /status, 2:5/0/64           ------> |
  |                                                        |
  | <------   ACK [MID=1238], 2.05 Content, 2:5/0/64       |
```

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).

```
CLIENT                                                   SERVER
   |                                                       |
   | CON [MID=1234], GET, /status                    ------> |
   |                                                       |
   | <------   ACK [MID=1234], 2.05 Content, 2:0/1/128     |
   |                                                       |
   | CON [MID=1235], GE//////////////////////              |
   |                                                       |
   | (timeout)                                             |
   |                                                       |
   | CON [MID=1235], GET, /status, 2:2/0/64          ------> |
   |                                                       |
   | <------   ACK [MID=1235], 2.05 Content, 2:2/1/64      |
   :                                                       :
   :                          ...                          :
   :                                                       :
   | CON [MID=1238], GET, /status, 2:5/0/64          ------> |
   |                                                       |
   | <------   ACK [MID=1238], 2.05 Content, 2:5/0/64      |
```

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

```
CLIENT                                                   SERVER
   |                                                       |
   | CON [MID=1234], GET, /status                    ------> |
   |                                                       |
   | <------   ACK [MID=1234], 2.05 Content, 2:0/1/128     |
   |                                                       |
   | CON [MID=1235], GET, /status, 2:2/0/64          ------> |
   |                                                       |
   | ////////////////////////////////tent, 2:2/1/64         |
   |                                                       |
   | (timeout)                                             |
   |                                                       |
   | CON [MID=1235], GET, /status, 2:2/0/64          ------> |
   |                                                       |
   | <------   ACK [MID=1235], 2.05 Content, 2:2/1/64      |
   :                                                       :
   :                          ...                          :
   :                                                       :
   | CON [MID=1238], GET, /status, 2:5/0/64          ------> |
   |                                                       |
   | <------   ACK [MID=1238], 2.05 Content, 2:5/0/64      |
```

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

3.2. Block1 Examples

The following examples demonstrate a PUT exchange; a POST exchange looks the same, with different requirements on atomicity/idempotence. Note that, similar to GET, the responses to the requests that have a more bit in the request Block1 Option are provisional and carry the response code 2.31 (Continue); only the final response tells the client that the PUT did succeed.

```
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 block-wise PUT

A stateless server that simply builds/updates the resource in place (statelessly) may indicate this by not setting the more bit in the response (Figure 8); in this case, the response codes are valid separately for each block being updated. This is of course only an acceptable behavior of the server if the potential inconsistency present during the run of the message exchange sequence does not lead to problems, e.g. because the resource being created or changed is not yet or not currently in use.

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

Figure 8: Simple stateless block-wise PUT

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

```
CLIENT                                                  SERVER
  |                                                       |
  | CON [MID=1234], PUT, /options, 1:0/1/128    ------>   |
  |                                                       |
  | <------   ACK [MID=1234], 2.31 Continue, 1:0/1/32     |
  |                                                       |
  | CON [MID=1235], PUT, /options, 1:4/1/32     ------>   |
  |                                                       |
  | <------   ACK [MID=1235], 2.31 Continue, 1:4/1/32     |
  |                                                       |
  | CON [MID=1236], PUT, /options, 1:5/1/32     ------>   |
  |                                                       |
  | <------   ACK [MID=1235], 2.31 Continue, 1:5/1/32     |
  |                                                       |
  | CON [MID=1237], PUT, /options, 1:6/0/32     ------>   |
  |                                                       |
  | <------   ACK [MID=1236], 2.04 Changed, 1:6/0/32      |
```

Figure 9: Simple atomic block-wise PUT with negotiation

3.3. Combining Block1 and Block2

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

```
CLIENT                                                    SERVER
  |                                                          |
  | CON [MID=1234], POST, /soap, 1:0/1/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1234], 2.31 Continue, 1:0/1/128       |
  |                                                          |
  | CON [MID=1235], POST, /soap, 1:1/1/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1235], 2.31 Continue, 1:1/1/128       |
  |                                                          |
  | CON [MID=1236], POST, /soap, 1:2/0/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1236], 2.04 Changed, 2:0/1/128, 1:2/0/128 |
  |                                                          |
  | CON [MID=1237], POST, /soap, 2:1/0/128      ------>      |
  | (no payload for requests with Block2 with NUM != 0)      |
  | (could also do late negotiation by requesting e.g. 2:2/0/64) |
  |                                                          |
  | <------   ACK [MID=1237], 2.04 Changed, 2:1/1/128        |
  |                                                          |
  | CON [MID=1238], POST, /soap, 2:2/0/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1238], 2.04 Changed, 2:2/1/128        |
  |                                                          |
  | CON [MID=1239], POST, /soap, 2:3/0/128      ------>      |
  |                                                          |
  | <------   ACK [MID=1239], 2.04 Changed, 2:3/0/128        |
```

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

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

```
   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

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
     |       |
     +------>|      Header: GET 0x41011636
     | GET   |       Token: 0xfb
     |       |    Uri-Path: status-icon
     |       |     Observe: (empty)
     |       |
     |<------+      Header: 2.05 0x61451636
```

```
         | 2.05 |      Token: 0xfb
         |      |     Block2: 0/1/128
         |      |    Observe: 62350
         |      |       ETag: 6f00f38e
         |      |    Payload: [128 bytes]
         |      |
         |      |  (Usual GET transfer left out)
           ...
         |      |  (Notification of first block:)
         |      |
         |<-----+     Header: 2.05 0x4145af9c
         | 2.05 |      Token: 0xfb
         |      |     Block2: 0/1/128
         |      |    Observe: 62354
         |      |       ETag: 6f00f392
         |      |    Payload: [128 bytes]
         |      |
         +- - ->|     Header: 0x6000af9c
         |      |
         |      |  (Retrieval of remaining blocks)
         |      |
         +----->|     Header: GET 0x41011637
         | GET  |      Token: 0xfc
         |      |   Uri-Path: status-icon
         |      |     Block2: 1/0/128
         |      |
         |<-----+     Header: 2.05 0x61451637
         | 2.05 |      Token: 0xfc
         |      |     Block2: 1/1/128
         |      |       ETag: 6f00f392
         |      |    Payload: [128 bytes]
         |      |
         +----->|     Header: GET 0x41011638
         | GET  |      Token: 0xfc
         |      |   Uri-Path: status-icon
         |      |     Block2: 2/0/128
         |      |
         |<-----+     Header: 2.05 0x61451638
         | 2.05 |      Token: 0xfc
         |      |     Block2: 2/0/128
         |      |       ETag: 6f00f392
         |      |    Payload: [53 bytes]
```

Figure 12: Observe sequence with block-wise response

(Note that the choice of token 0xfc in this examples is arbitrary; tokens are just shown in this example to illustrate that the requests

for additional blocks cannot make use of the token of the Observation relationship. As a general comment on tokens, there is no other mention of tokens in this document, as block-wise transfers handle tokens like any other CoAP exchange. As usual the client is free to choose tokens for each exchange as it likes.)

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

```
CLIENT  SERVER
  |       |
  +------>|     Header: GET 0x41011636
  | GET   |      Token: 0xfb
  |       |   Uri-Path: status-icon
  |       |    Observe: (empty)
  |       |     Block2: 0/0/64
  |       |
  |<------+     Header: 2.05 0x61451636
  | 2.05  |      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:)
  |       |
  |<------+     Header: 2.05 0x4145af9c
  | 2.05  |      Token: 0xfb
  |       |     Block2: 0/1/64
  |       |    Observe: 62354
  |       |       ETag: 6f00f392
  |       |    Payload: [64 bytes]
  |       |
  +- - ->|     Header: 0x6000af9c
  |       |
  |       |  (Retrieval of remaining blocks)
  |       |
  +------>|     Header: GET 0x41011637
  | GET   |      Token: 0xfc
  |       |   Uri-Path: status-icon
  |       |     Block2: 1/0/64
  |       |
  |<------+     Header: 2.05 0x61451637
  | 2.05  |      Token: 0xfc
  |       |     Block2: 1/1/64
```

```
         |       |       ETag: 6f00f392
         |       |    Payload: [64 bytes]
           ....
         |       |
         +------>|     Header: GET 0x41011638
         | GET   |      Token: 0xfc
         |       |   Uri-Path: status-icon
         |       |     Block2: 4/0/64
         |       |
         |<------+     Header: 2.05 0x61451638
         | 2.05  |      Token: 0xfc
         |       |     Block2: 4/0/64
         |       |       ETag: 6f00f392
         |       |    Payload: [53 bytes]
```

Figure 13: Observe sequence with early negotiation

4. The Size2 and Size1 Options

In many cases when transferring a large resource representation block by block, it is advantageous to know the total size early in the process. Some indication may be available from the maximum size estimate attribute "sz" provided in a resource description [RFC6690]. However, the size may vary dynamically, so a more up-to-date indication may be useful.

This specification defines two CoAP Options, Size1 for indicating the size of the representation transferred in requests, and Size2 for indicating the size of the representation transferred in responses. (Size1 has already been defined in Section 5.10.9 of [RFC7252] to provide "size information about the resource representation in a request", however that section only details the narrow case of indicating in 4.13 responses the maximum size of request payload that the server is able and willing to handle. The present specification provides details about its use as a request option as well.)

The Size2 Option may be used for two purposes:

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

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

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

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

- o in a 4.13 response, to indicate the maximum size that would have been acceptable [RFC7252], measured in bytes.

Apart from conveying/asking for size information, the Size options have no other effect on the processing of the request or response. If the client wants to minimize the size of the payload in the resulting response, it should add a Block2 option to the request with a small block size (e.g., setting SZX=0).

The Size Options are "elective", i.e., a client MUST be prepared for the server to ignore the size estimate request. The Size Options MUST NOT occur more than once.

No.	C	U	N	R	Name	Format	Length	Default
60			x		Size1	uint	0-4	(none)
28			x		Size2	uint	0-4	(none)

Table 2: Size Option Numbers

Implementation Notes:

- o As a quality of implementation consideration, block-wise transfers for which the total size considerably exceeds the size of one block are expected to include size indications, whenever those can be provided without undue effort (preferably with the first block exchanged). If the size estimate does not change, the indication does not need to be repeated for every block.

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

- o As usual for an option of type uint, the value 0 is best expressed as an empty option (0 bytes). There is no default value for either Size Option.

- o The Size Options are neither critical nor unsafe, and are marked as No-Cache-Key.

5. HTTP Mapping Considerations

In this subsection, we give some brief examples for the influence the Block options might have on intermediaries that map between CoAP and HTTP.

For mapping CoAP requests to HTTP, the intermediary may want to map the sequence of block-wise transfers into a single HTTP transfer. E.g., for a GET request, the intermediary could perform the HTTP request once the first block has been requested and could then fulfill all further block requests out of its cache. A constrained implementation may not be able to cache the entire object and may use a combination of TCP flow control and (in particular if timeouts occur) HTTP range requests to obtain the information necessary for the next block transfer at the right time.

For PUT or POST requests, historically there was more variation in how HTTP servers might implement ranges; recently, [RFC7233] has defined that Range header fields received with a request method other than GET are not to be interpreted. So, in general, the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer for these other methods within one HTTP request. If enough buffering is available, this request can be started when the last CoAP block is received. A constrained implementation may want to relieve its buffering by already starting to send the HTTP request at the time the first CoAP block is received; any HTTP 408 status code that indicates that the HTTP server became impatient with the resulting transfer can then be mapped into a CoAP 4.08 response code (similarly, 413 maps to 4.13).

For mapping HTTP to CoAP, the intermediary may want to map a single HTTP transfer into a sequence of block-wise transfers. If the HTTP client is too slow delivering a request body on a PUT or POST, the CoAP server might time out and return a 4.08 response code, which in turn maps well to an HTTP 408 status code (again, 4.13 maps to 413). HTTP range requests received on the HTTP side may be served out of a cache and/or mapped to GET requests that request a sequence of blocks overlapping the range.

(Note that, while the semantics of CoAP 4.08 and HTTP 408 differ, this difference is largely due to the different way the two protocols are mapped to transport. HTTP has an underlying TCP connection, which supplies connection state, so a HTTP 408 status code can immediately be used to indicate that a timeout occurred during transmitting a request through that active TCP connection. The CoAP 4.08 response code indicates one or more missing blocks, which may be due to timeouts or resource constraints; as there is no connection state, there is no way to deliver such a response immediately;

instead, it is delivered on the next block transfer. Still, HTTP 408 is probably the best mapping back to HTTP, as the timeout is the most likely cause for a CoAP 4.08. Note that there is no way to distinguish a timeout from a missing block for a server without creating additional state, the need for which we want to avoid.)

6. IANA Considerations

This draft adds the following option numbers to the CoAP Option Numbers registry of [RFC7252]:

Number	Name	Reference
23	Block2	[RFCXXXX]
27	Block1	[RFCXXXX]
28	Size2	[RFCXXXX]

Table 3: CoAP Option Numbers

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

Code	Description	Reference
2.31	Continue	[RFCXXXX]
4.08	Request Entity Incomplete	[RFCXXXX]

Table 4: CoAP Response Codes

7. Security Considerations

Providing access to blocks within a resource may lead to surprising vulnerabilities. Where requests are not implemented atomically, an attacker may be able to exploit a race condition or confuse a server by inducing it to use a partially updated resource representation. Partial transfers may also make certain problematic data invisible to intrusion detection systems; it is RECOMMENDED that an intrusion detection system (IDS) that analyzes resource representations transferred by CoAP implement the Block options to gain access to entire resource representations. Still, approaches such as transferring even-numbered blocks on one path and odd-numbered blocks

on another path, or even transferring blocks multiple times with different content and obtaining a different interpretation of temporal order at the IDS than at the server, may prevent an IDS from seeing the whole picture. These kinds of attacks are well understood from IP fragmentation and TCP segmentation; CoAP does not add fundamentally new considerations.

Where access to a resource is only granted to clients making use of specific security associations, all blocks of that resource MUST be subject to the same security checks; it MUST NOT be possible for unprotected exchanges to influence blocks of an otherwise protected resource. As a related consideration, where object security is employed, PUT/POST should be implemented in the atomic fashion, unless the object security operation is performed on each access and the creation of unusable resources can be tolerated. Future end-to-end security mechanisms that may be added to CoAP itself may have related security considerations, this includes considerations about caching of blocks in clients and in proxies (see Section 2.10 and Section 5 for different strategies in performing this caching); these security considerations will need to be described in the specifications of those mechanisms.

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

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

7.1. Mitigating Resource Exhaustion Attacks

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

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

7.2. Mitigating Amplification Attacks

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

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

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

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

8.2. Informative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <http://www.rfc-editor.org/info/rfc4919>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <http://www.rfc-editor.org/info/rfc4944>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <http://www.rfc-editor.org/info/rfc7233>.

Acknowledgements

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

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

insisted on a more systematic coverage of the options and response code. Qin Wu provided a review for the IETF Operational directorate, and Goeran Selander commented on the security considerations.

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

The IESG reviewers provided very useful comments. Spencer Dawkins even suggested new text. Mirja Kuehlewind and he insisted on being more explicit about the layering of block-wise transfers on top of the base protocol. Ben Campbell helped untangling some MUST/SHOULD soup. Comments by Alexey Melnikov, as well as the gen-art review by Jouni Korhonen and the ops-dir review by Qin Wu, caused further improvements to the text.

Authors' Addresses

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

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

Zach Shelby (editor)
ARM
150 Rose Orchard
San Jose, CA 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 1, 2017

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
Nokia
E. Dijk
Philips Lighting
November 28, 2016

Guidelines for HTTP-to-CoAP Mapping Implementations
draft-ietf-core-http-mapping-17

Abstract

This document provides reference information for implementing a cross-protocol network proxy that performs translation from the HTTP protocol to CoAP (Constrained Application Protocol). This will enable an HTTP client to access resources on a CoAP server through the proxy. This document describes how an HTTP request is mapped to a CoAP request, and then how a CoAP response is mapped back to an HTTP response. This includes guidelines for status code, URI, and media type mappings, as well as additional interworking advice.

Copyright Notice

Table of Contents

1. Introduction

CoAP (Constrained Application Protocol) [RFC7252] has been designed with the twofold aim to be an application protocol specialized for constrained environments and to be easily used in Representational State Transfer (REST) [Fielding] based architectures such as the Web. The latter goal has led to defining CoAP to easily interoperate with HTTP [RFC7230] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [RFC7252] describes the fundamentals of the CoAP-to-HTTP and the HTTP-to-CoAP cross-protocol mapping process. However, [RFC7252] focuses on the basic mapping of request methods and simple response code mapping between HTTP and CoAP, while leaving many details of the cross-protocol proxy for future definition. Therefore, a primary goal of this document is to define a consistent set of guidelines that an HTTP-to-CoAP proxy implementation should adhere to. The key benefit to adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability between an HTTP client and a CoAP server independent of the proxy that implements the cross-protocol mapping. (For example, a proxy conforming to these guidelines made by vendor A can be easily replaced by a proxy from vendor B that also conforms to the guidelines without breaking API semantics.)

This document describes HTTP mappings that apply to protocol elements defined in the base CoAP specification [RFC7252]. It is up to CoAP protocol extensions (new methods, response codes, options, content-formats) to describe their own HTTP mappings, if applicable.

The rest of this document is organized as follows:

- Section 2 defines proxy terminology;
- Section 3 introduces the HTTP-to-CoAP proxy;
- Section 4 lists use cases in which HTTP clients need to contact CoAP servers;
- Section 5 introduces a null, default, and advanced HTTP-to-CoAP URI mapping syntax;
- Section 6 describes how to map HTTP media types to CoAP content formats and vice versa;
- Section 7 describes how to map CoAP responses to HTTP responses;
- Section 8 describes additional mapping guidelines related to caching, congestion, timeouts, etc.;
- Section 10 discusses possible security impact of HTTP-to-CoAP protocol mapping.

2. Terminology

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

This specification requires readers to be familiar with the vocabulary and concepts discussed in [RFC7228], in particular, the terms "Constrained Nodes" and "Constrained Networks". In addition, this specification makes use of the following terms:

HC Proxy
: A proxy performing a cross-protocol mapping, in the context of this document an HTTP-to-CoAP (HC) mapping. Specifically, the HC Proxy acts as an HTTP server and a CoAP client. The HC Proxy can take on the role of a Forward, Reverse or Interception Proxy.

Forward Proxy (or Forward HC Proxy)
: A message forwarding agent that is selected by the HTTP client, usually via local configuration rules, to receive requests for some type(s) of absolute URI and to attempt to satisfy those requests via translation to the protocol indicated by the absolute URI. The user decides (is willing) to use the proxy as the forwarding/de-referencing agent for a predefined subset of the URI space. In [RFC7230] this is called a Proxy. [RFC7252] defines Forward-Proxy similarly.

Reverse Proxy (or Reverse HC Proxy)
: As in [RFC7230], a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server's protocol. A Reverse HC Proxy behaves as an origin (HTTP) server on its connection from the HTTP client. The HTTP client uses the "origin-form" (Section 5.3.1 of [RFC7230]) as a request-target URI. (Note that a Reverse Proxy appears to an HTTP client as an origin server while a Forward Proxy does not. So, when communicating with a Reverse Proxy a client may be unaware it is communicating with a proxy at all.)

Interception Proxy (or Interception HC Proxy)
: As in [RFC3040], a proxy that receives inbound HTTP traffic flows through the process of traffic redirection, transparent to the HTTP client.

3. HTTP-to-CoAP Proxy

An HC Proxy is accessed by an HTTP client that needs to fetch a resource on a CoAP server. The HC Proxy handles the HTTP request by mapping it to the equivalent CoAP request, which is then forwarded to the appropriate CoAP server. The received CoAP response is then mapped to an appropriate HTTP response and finally sent back to the originating HTTP client.

Section 10.2 of [RFC7252] defines basic normative requirements on HTTP-to-CoAP mapping. This document provides additional details and guidelines for the implementation of an HC Proxy.

```
                                                Constrained Network
                                              .-------------------.
                                             /      .------.       \
                                            /       | CoAP |        \
                                           /        |server|         \
                                          ||        '------'         ||
                                          ||                         ||
      .--------.  HTTP Request   .------------.  CoAP Req  .------.  ||
      |  HTTP  |---------------->|HTTP-to-CoAP|----------->| CoAP |  ||
      | Client |<----------------|   Proxy    |<-----------|Server|  ||
      '--------'  HTTP Response  '------------'  CoAP Resp '------'  ||
                                          ||                         ||
                                          ||   .------.             ||
                                          ||   | CoAP |             ||
                                           \   |server|  .------.    /
                                            \  '------'  | CoAP |   /
                                             \           |server|  /
                                              \          '------' /
                                               '-----------------'
```

Figure 1: HTTP-To-CoAP Proxy Deployment Scenario

Figure 1 illustrates an example deployment scenario. There, an HC Proxy is located at the boundary of the Constrained Network domain and acts as an Application Layer Gateway (ALG) that allows only a very specific type of traffic (i.e., authorized inbound HTTP requests and their associated outbound CoAP responses) to pass through. All other kinds of traffic are segregated within the respective network segments.

4. Use Cases

To illustrate a few situations in which HTTP to CoAP protocol translation may be used, three use cases are described below.

1. Legacy building control application without CoAP: A building control application that uses HTTP but not CoAP can check the status of CoAP sensors and/or control actuators via an HC Proxy.

2. Making sensor data available to 3rd parties on the Web: For demonstration or public interest purposes, an HC Proxy may be configured to expose the contents of a CoAP sensor to the world via the web (HTTP and/or HTTPS). Some sensors may only accept secure 'coaps' requests, therefore the proxy is configured to translate requests to those devices accordingly. The HC Proxy is furthermore configured to only pass through GET requests in order to protect the constrained network.

3. Smartphone and home sensor: A smartphone can access directly a CoAP home sensor using a mutually authenticated 'https' request, provided its home router runs an HC Proxy and is configured with the appropriate certificate. An HTML5 [W3C.REC-html5-20141028] application on the smartphone can provide a friendly UI using the standard (HTTP) networking functions of HTML5.

A key point in the above use cases is the expected nature of the URI to be used by the HTTP client initiating the HTTP request to the HC Proxy. Specifically, in use case #1, there will be no 'coap' or 'coaps' related information embedded in the HTTP URI as it is a legacy HTTP client sending the request. Use case #2 is also expected to be similar. In contrast, in use case #3, it is likely that the HTTP client will specifically embed 'coap' or 'coaps' related information in the HTTP URI of the HTTP request to the HC Proxy.

5. URI Mapping

Though, in principle, a CoAP URI could be directly used by an HTTP client to de-reference a CoAP resource through an HC Proxy, the reality is that all major web browsers, networking libraries and command line tools do not allow making HTTP requests using URIs with a scheme 'coap' or 'coaps'.

Thus, there is a need for web applications to embed or "pack" a CoAP URI into an HTTP URI so that it can be (non-destructively) transported from the HTTP client to the HC Proxy. The HC Proxy can then "unpack" the CoAP URI and finally de-reference it via a CoAP request to the target Server.

URI Mapping is the term used in this document to describe the process through which the URI of a CoAP resource is transformed into an HTTP URI so that:

- The requesting HTTP client 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 an 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 learn about

the supported URI Mapping Template(s), as well as the URI where the HC Proxy function is anchored.

5.1. URI Terminology

In the remainder of this section, the following terms will be used with a distinctive meaning:

HC Proxy URI:
URI which refers to the HC Proxy function. It conforms to syntax defined in Section 2.7 of [RFC7230].

Target CoAP URI:
URI which refers to the (final) CoAP resource that has to be de-referenced. It conforms to syntax defined in Section 6 of [RFC7252]. Specifically, its scheme is either 'coap' or 'coaps'.

Hosting HTTP URI:
URI that conforms to syntax in Section 2.7 of [RFC7230]. Its authority component refers to an HC Proxy, whereas path and/or query component(s) embed the information used by an HC Proxy to extract the Target CoAP URI.

5.2. Null Mapping

The null mapping is the case where there is no Target CoAP URI appended to the HC Proxy URI. In other words, it is a "pure" HTTP URI that is sent to the HC Proxy. This would typically occur in situations like Use Case #1 described in Section 4, and the Proxy would typically be a Reverse Proxy. In this scenario, the HC Proxy will determine through its own private algorithms what the Target CoAP URI should be.

5.3. Default Mapping

The default mapping is for the Target CoAP URI to be appended as-is (with the only caveat discussed in Section 5.3.2) to the HC Proxy URI, to form the Hosting HTTP URI. This is the Effective Request URI (see Section 5.5 of [RFC7230]) that will then be sent by the HTTP client in the HTTP request to the HC Proxy.

For example: given an HC Proxy URI https://p.example.com/hc/ and a Target CoAP URI coap://s.example.com/light, the resulting Hosting HTTP URI would be https://p.example.com/hc/coap://s.example.com/light.

Provided a correct Target CoAP URI, the Hosting HTTP URI resulting from the default mapping will be a syntactically valid HTTP URI. Furthermore, the Target CoAP URI can always be extracted unambiguously from the Hosting HTTP URI.

There is no default for the HC Proxy URI. Therefore, it is either known in advance, e.g., as a configuration preset, or dynamically discovered using the mechanism described in Section 5.5.

The default URI mapping function SHOULD be implemented and SHOULD be activated by default in an HC Proxy, unless there are valid reasons (e.g., application specific) to use a different mapping function.

5.3.1. Optional Scheme Omission

When constructing a Hosting HTTP URI by embedding a Target CoAP URI, the scheme (i.e., 'coap' or 'coaps'), the scheme component delimiter (":"), and the double slash ("//") preceding the authority MAY be omitted if a local default - not defined by this document - applies. If no prior mutual agreement exists between the client and the HC Proxy, then a Target CoAP URI without the scheme component is syntactically incorrect, and therefore:

- It MUST NOT be emitted by clients;
- It MUST elicit a suitable client error status (i.e., 4xx) by the HC Proxy.

5.3.2. Encoding Caveats

When the authority of the Target CoAP URI is given as an IPv6address, then the surrounding square brackets must be percent-encoded in the Hosting HTTP URI, in order to comply with the syntax defined in Section 3.3. of [RFC3986] for a URI path segment. E.g.: coap://[2001:db8::1]/light?on becomes https://p.example.com/hc/coap://%5B2001:db8::1%5D/light?on. (Note that the percent-encoded square brackets shall be reverted to their non-percent-encoded form when the HC Proxy unpacks the Target CoAP URI.)

Everything else can be safely copied verbatim from the Target CoAP URI to the Hosting HTTP URI.

5.4. URI Mapping Template

This section defines a format for the URI template [RFC6570] used by an HC Proxy to inform its clients about the expected syntax for the Hosting HTTP URI. This will then be used by the HTTP client to

construct the Effective Request URI to be sent in the HTTP request to the HC Proxy.

When instantiated, a URI Mapping Template is always concatenated to an HC Proxy URI provided by the HC Proxy via discovery (see Section 5.5), or by other means.

A simple form (Section 5.4.1) and an enhanced form (Section 5.4.2) are provided to fit different users' requirements.

Both forms are expressed as level 2 URI templates [RFC6570] to take care of the expansion of values that are allowed to include reserved URI characters. The syntax of all URI formats is specified in this section in Augmented Backus-Naur Form (ABNF) [RFC5234].

5.4.1. Simple Form

The simple form MUST be used for mappings where the Target CoAP URI is going to be copied (using rules of Section 5.3.2) at some fixed position into the Hosting HTTP URI.

The "tu" template variable is intended to be used in a template definition to represent a Target CoAP URI:

```
tu = [ ( "coap:" / "coaps:" ) "//" ] host [ ":" port ] path-abempty
     [ "?" query ]
```

Note that the same considerations as in Section 5.3.1 apply, in that the CoAP scheme may be omitted from the Hosting HTTP URI.

5.4.1.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use https://p.example.com/hc/ as the HC Proxy URI. Note that these examples all define mapping templates that deviate from the default template of Section 5.3 in order to illustrate the use of the above template variables.

1. Target CoAP URI is a query argument of the Hosting HTTP URI:

```
?target_uri={+tu}

coap://s.example.com/light

=> https://p.example.com/hc/?target_uri=coap://s.example.com/light

whereas

coaps://s.example.com/light

=> https://p.example.com/hc/?target_uri=coaps://s.example.com/light
```

2. Target CoAP URI in the path component of the Hosting HTTP URI:

```
forward/{+tu}

coap://s.example.com/light

=> https://p.example.com/hc/forward/coap://s.example.com/light

whereas

coaps://s.example.com/light

=> https://p.example.com/hc/forward/coaps://s.example.com/light
```

3. Target CoAP URI is a query argument of the Hosting HTTP URI; client decides to omit the scheme because a default is agreed beforehand between client and proxy:

```
?coap_uri={+tu}

coap://s.example.com/light

=> https://p.example.com/hc/?coap_uri=s.example.com/light
```

5.4.2. Enhanced Form

The enhanced form can be used to express more sophisticated mappings of the Target CoAP URI into the Hosting HTTP URI, i.e., mappings that do not fit into the simple form.

There MUST be at most one instance of each of the following template variables in a template definition:

```
s  = "coap" / "coaps" ; from [RFC7252], Sections 6.1 and 6.2
hp = host [":" port]  ; from [RFC3986], Sections 3.2.2 and 3.2.3
p  = path-abempty     ; from [RFC3986], Section 3.3
q  = query            ; from [RFC3986], Section 3.4
qq = [ "?" query ]    ; qq is empty if and only if 'query' is empty
```

The qq form is used when the path and the (optional) query components are to be copied verbatim from the Target CoAP URI into the Hosting HTTP URI, i.e., as "{+p}{+qq}". Instead, the q form is used when the query and path are mapped as separate entities, e.g., as in "coap_path={+p}&coap_query={+q}".

5.4.2.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use https://p.example.com/hc/ as the HC Proxy URI.

1. Target CoAP URI components in path segments, and optional query in query component:

   {+s}/{+hp}{+p}{+qq}

   coap://s.example.com/light

   => https://p.example.com/hc/coap/s.example.com/light

   whereas

   coap://s.example.com/light?on

   => https://p.example.com/hc/coap/s.example.com/light?on

2. Target CoAP URI components split in individual query arguments:

```
?s={+s}&hp={+hp}&p={+p}&q={+q}

coap://s.example.com/light

=> https://p.example.com/hc/?s=coap&hp=s.example.com&p=/light&q=

whereas

coaps://s.example.com/light?on

=> https://p.example.com/hc/?s=coaps&hp=s.example.com&p=/light&q=on
```

5.5. Discovery

In order to accommodate site-specific needs while allowing third parties to discover the proxy function, the HC Proxy SHOULD publish information related to the location and syntax of the HC Proxy function using the CoRE Link Format [RFC6690] interface.

To this aim a new Resource Type, "core.hc", is defined in this document. It can be used as the value for the "rt" attribute in a query to the /.well-known/core in order to locate the URI where the HC Proxy function is anchored, i.e., the HC Proxy URI.

Along with it, the new target attribute "hct" is defined in this document. This attribute MAY be returned in a "core.hc" link to provide the URI Mapping Template associated with the mapping resource. The default template given in Section 5.3, i.e., {+tu}, MUST be assumed if no "hct" attribute is found in the returned link. If a "hct" attribute is present in the returned link, then a client MUST use it to create the Hosting HTTP URI.

The URI mapping SHOULD be discoverable (as specified in [RFC6690]) on both the HTTP and the CoAP side of the HC Proxy, with one important difference: on the CoAP side the link associated with the "core.hc" resource needs an explicit anchor referring to the HTTP origin [RFC6454], while on the HTTP interface the link context is already the HTTP origin carried in the request’s Host header, and doesn’t have to be made explicit.

5.5.1. Examples

- The first example exercises the CoAP interface and assumes that the default template, {+tu}, is used. For example, a smartphone may discover the public HC Proxy before leaving the home network. Then when outside the home network, the smartphone will be able to query the appropriate home sensor.

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

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

- The second example - also on the CoAP side of the HC Proxy - uses a custom template, i.e., one where the CoAP URI is carried inside the query component, thus the returned link carries the URI template to be used in an explicit "hct" attribute:

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

Res:  2.05 Content
      </hc/>;anchor="https://p.example.com";
      rt="core.hc";hct="?uri={+tu}"
```

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

- 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.ietf-core-links-json]:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format+json
      Content-Length: 31

      [{"href":"/hc/","rt":"core.hc"}]
```

o using the Link header:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com

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

6. Media Type Mapping

6.1. Overview

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

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

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

On the content negotiation side, failure to map Accept and Accept-* headers SHOULD be silently ignored: the HC Proxy SHOULD therefore forward as a CoAP request with no Accept option. The HC Proxy thus disregards the Accept/Accept-* header fields by treating the response as if it is not subject to content negotiation, as mentioned in Sections 5.3.* of [RFC7231]. However, an HC Proxy implementation is free to attempt mapping a single Accept header in a GET request to multiple CoAP GET requests, each with a single Accept option, which are then tried in sequence until one succeeds. Note that an HTTP Accept */* MUST be mapped to a CoAP request without Accept option.

While the CoAP to HTTP direction has always a well-defined mapping (with the exception examined in Section 6.2), the HTTP to CoAP direction is more problematic because the source set, i.e., potentially 1000+ IANA registered media types, is much bigger than

the destination set, i.e., the mere 6 values initially defined in Section 12.3 of [RFC7252].

Depending on the tight/loose coupling with the application(s) for which it proxies, the HC Proxy could implement different media type mappings.

When tightly coupled, the HC Proxy knows exactly which content formats are supported by the applications, and can be strict when enforcing its forwarding policies in general, and the media type mapping in particular.

On the other hand, when the HC Proxy is a general purpose ALG, being too strict could significantly reduce the amount of traffic that it would be able to successfully forward. In this case, the "loose" media type mapping detailed in Section 6.3 MAY be implemented.

The latter grants more evolution of the surrounding ecosystem, at the cost of allowing more attack surface. In fact, as a result of such strategy, payloads would be forwarded more liberally across the unconstrained/constrained network boundary of the communication path.

6.2. 'application/coap-payload' Media Type

If the HC Proxy receives a CoAP response with a Content-Format that it does not recognize (e.g., because the value has been registered after the proxy has been deployed, or the CoAP server uses an experimental value which is not registered), then the HC Proxy SHALL return a generic "application/coap-payload" media type with numeric parameter "cf" as defined in Section 9.2.

For example, the CoAP content format '60' ("application/cbor") would be represented by "application/coap-payload;cf=60", if the HC Proxy doesn't recognize the content format '60'.

A HTTP client may use the media type "application/coap-payload" as a means to send a specific content format to a CoAP server via an HC Proxy if the client has determined that the HC Proxy does not directly support the type mapping it needs. This case may happen when dealing for example with newly registered, yet to be registered, or experimental CoAP content formats. However, unless explicitly configured to allow pass-through of unknown content formats, the HC Proxy SHOULD NOT forward requests carrying a Content-Type or Accept header with an "application/coap-payload", and return an appropriate client error instead.

6.3. Loose Media Type Mapping

By structuring the type information in a super-class (e.g., "text") followed by a finer grained sub-class (e.g., "html"), and optional parameters (e.g., "charset=utf-8"), Internet media types provide a rich and scalable framework for encoding the type of any given entity.

This approach is not applicable to CoAP, where Content Formats conflate an Internet media type (potentially with specific parameters) and a content encoding into one small integer value.

To remedy this loss of flexibility, we introduce the concept of a "loose" media type mapping, where media types that are specializations of a more generic media type can be aliased to their super-class and then mapped (if possible) to one of the CoAP content formats. For example, "application/soap+xml" can be aliased to "application/xml", which has a known conversion to CoAP. In the context of this "loose" media type mapping, "application/octet-stream" can be used as a fallback when no better alias is found for a specific media type.

Table 1 defines the default lookup table for the "loose" media type mapping. It is expected that an implementation can refine it either given application-specific knowledge, or because new Content-Formats are defined. Given an input media type, the table returns its best generalized media type using the most specific match i.e., the table entries are compared to the input in top to bottom order until an entry matches.

Internet media type pattern	Generalized media type
application/*+xml	application/xml
application/*+json	application/json
application/*+cbor	application/cbor
text/xml	application/xml
text/*	text/plain
/	application/octet-stream

Table 1: Media type generalization lookup table

The "loose" media type mapping is an OPTIONAL feature. Implementations supporting this kind of mapping should provide a flexible way to define the set of media type generalizations allowed.

6.4. Media Type to Content Format Mapping Algorithm

This section defines the algorithm used to map an HTTP Internet media type to its correspondent CoAP content format; it can be used as a building block for translating HTTP Content-Type and Accept headers into CoAP Content-Format and Accept Options.

The algorithm uses an IANA-maintained table, "CoAP Content-Formats", as established by Section 12.3 of [RFC7252] plus, possibly, any locally defined extension of it. Optionally, the table and lookup mechanism described in Section 6.3 can be used if the implementation chooses so.

Note that the algorithm assumes an "identity" Content-Encoding and expects the resource body has been already successfully content-decoded or transcoded to the desired format.

In the following (Figure 2):

- o media_type is the media type to translate;
- o coap_cf_registry is a lookup table matching the CoAP Content Format Registry;
- o loose_mapper is an optional lookup table describing the loose media type mappings (e.g., the one defined in Table 1);

The full source code is provided in Appendix A.

```
def mt2cf(media_type, encoding=None,
          coap_cf_registry=CoAPContentFormatRegistry(),
          loose_mapper=None):
    """Return a CoAP Content-Format given an Internet Media Type and
       its optional encoding.  The current (as of 2016/10/24) CoAP
       Content Format Registry is supplied by default.  An optional
       'loose-mapping' implementation can be supplied by the caller."""
    assert media_type is not None
    assert coap_cf_registry is not None

    # Lookup the CoAP Content-Formats registry
    content_format = coap_cf_registry.lookup(media_type, encoding)

    # If an exact match is not found and a loose mapper has been
    # supplied, try to use it to get a media type with which to
    # re-try the CoAP Content-Formats registry lookup.
    if content_format is None and loose_mapper is not None:
        content_format = coap_cf_registry.lookup(
            loose_mapper.lookup(media_type), encoding)

    return content_format
```

Figure 2

6.5. Content Transcoding

6.5.1. General

Payload content transcoding is an OPTIONAL feature. Implementations supporting this feature should provide a flexible way to define the set of transcodings allowed.

The HC Proxy might decide to transcode the received representation to a different (compatible) format when an optimized version of a specific format exists. For example, a XML-encoded resource could be transcoded to Efficient XML Interchange (EXI) format, or a JSON-encoded resource into CBOR [RFC7049], effectively achieving compression without losing any information.

However, there are a few important factors to keep in mind when enabling a transcoding function:

1. Maliciously crafted inputs coming from the HTTP side might inflate in size (see for example Section 4.2 of [RFC7049]), therefore creating a security threat for both the HC Proxy and the target resource;

2. Transcoding can lose information in non-obvious ways. For example, encoding a XML document using schema-informed EXI encoding leads to a loss of information when the destination does not know the exact schema version used by the encoder. That means that whenever the HC Proxy transcodes an application/XML to application/EXI in-band metadata could be lost.

3. When content-type is mapped, there is a risk that the content with the destination type would have malware not active in the source type.

It is crucial that these risks are well understood and carefully weighed against the actual benefits before deploying the transcoding function.

6.5.2. CoRE Link Format

The CoRE Link Format [RFC6690] is a set of links (i.e., URIs and their formal relationships) which is carried as content payload in a CoAP response. These links usually include CoAP URIs that might be translated by the HC Proxy to the correspondent HTTP URIs using the implemented URI mapping function (see Section 5). Such a process would inspect the forwarded traffic and attempt to re-write the body of resources with an application/link-format media type, mapping the embedded CoAP URIs to their HTTP counterparts. Some potential issues with this approach are:

1. The client may be interested in retrieving original (unaltered) CoAP payloads through the HC Proxy, not modified versions.

2. Tampering with payloads is incompatible with resources that are integrity protected (although this is a problem with transcoding in general).

3. The HC Proxy needs to fully understand [RFC6690] syntax and semantics, otherwise there is an inherent risk to corrupt the payloads.

Therefore, CoRE Link Format payload should only be transcoded at the risk and discretion of the proxy implementer.

6.5.3. Diagnostic Messages

CoAP responses may, in certain error cases, contain a diagnostic message in the payload explaining the error situation, as described in Section 5.5.2 of [RFC7252]. If present, the CoAP response diagnostic payload SHOULD be copied in the HTTP response body. The CoAP diagnostic message MUST NOT be copied into the HTTP reason-

phrase, since it potentially contains CR-LF characters which are incompatible with HTTP reason-phrase syntax.

7. Response Code Mapping

Table 2 defines the HTTP response status codes to which each CoAP response code SHOULD be mapped. Multiple appearances of a HTTP status code in the second column indicates multiple equivalent HTTP responses are possible based on the same CoAP response code, depending on the conditions cited in the Notes (third column and text below table).

CoAP Response Code	HTTP Status Code	Note
2.01 Created	201 Created	1
2.02 Deleted	200 OK	2
	204 No Content	2
2.03 Valid	304 Not Modified	3
	200 OK	4
2.04 Changed	200 OK	2
	204 No Content	2
2.05 Content	200 OK	
2.31 Continue	N/A	10
4.00 Bad Request	400 Bad Request	
4.01 Unauthorized	403 Forbidden	5
4.02 Bad Option	400 Bad Request	6
4.02 Bad Option	500 Internal Server Error	6
4.03 Forbidden	403 Forbidden	
4.04 Not Found	404 Not Found	
4.05 Method Not Allowed	400 Bad Request	7
4.06 Not Acceptable	406 Not Acceptable	
4.08 Request Entity Incomplt.	N/A	10
4.12 Precondition Failed	412 Precondition Failed	
4.13 Request Ent. Too Large	413 Payload Too Large	11
4.15 Unsupported Content-Fmt.	415 Unsupported Media Type	
5.00 Internal Server Error	500 Internal Server Error	
5.01 Not Implemented	501 Not Implemented	
5.02 Bad Gateway	502 Bad Gateway	
5.03 Service Unavailable	503 Service Unavailable	8
5.04 Gateway Timeout	504 Gateway Timeout	
5.05 Proxying Not Supported	502 Bad Gateway	9

Table 2: CoAP-HTTP Response Code Mappings

Notes:

1. A CoAP server may return an arbitrary format payload along with this response. If present, this payload MUST be returned as entity in the HTTP 201 response. Section 7.3.2 of [RFC7231] does not put any requirement on the format of the entity. (In the past, [RFC2616] did.)

2. The HTTP code is 200 or 204 respectively for the case that a CoAP server returns a payload or not. [RFC7231] Section 5.3 requires code 200 in case a representation of the action result is returned for DELETE/POST/PUT, and code 204 if not. Hence, a proxy MUST transfer any CoAP payload contained in a CoAP 2.02 response to the HTTP client using a 200 OK response.

3. HTTP code 304 (Not Modified) is sent if the HTTP client performed a conditional HTTP request and the CoAP server responded with 2.03 (Valid) to the corresponding CoAP validation request. Note that Section 4.1 of [RFC7232] puts some requirements on header fields that must be present in the HTTP 304 response.

4. A 200 response to a CoAP 2.03 occurs only when the HC Proxy, for efficiency reasons, is running a local cache. An unconditional HTTP GET which produces a cache-hit, could trigger a re-validation (i.e., a conditional GET) on the CoAP side. The proxy receiving 2.03 updates the freshness of its cached representation and returns it to the HTTP client.

5. A HTTP 401 Unauthorized (Section 3.1 of [RFC7235]) response is not applicable because there is no equivalent in CoAP of WWW-Authenticate which is mandatory in a HTTP 401 response.

6. If the proxy has a way to determine that the Bad Option is due to the straightforward mapping of a client request header into a CoAP option, then returning HTTP 400 (Bad Request) is appropriate. In all other cases, the proxy MUST return HTTP 500 (Internal Server Error) stating its inability to provide a suitable translation to the client’s request.

7. A CoAP 4.05 (Method Not Allowed) response SHOULD normally be mapped to a HTTP 400 (Bad Request) code, because the HTTP 405 response would require specifying the supported methods - which are generally unknown. In this case the HC Proxy SHOULD also return a HTTP reason-phrase in the HTTP status line that starts with the string "CoAP server returned 4.05" in order to facilitate troubleshooting. However, if the HC Proxy has more granular information about the supported methods for the requested resource (e.g., via a Resource Directory ([I-D.ietf-core-resource-directory])) then it MAY send back a

HTTP 405 (Method Not Allowed) with a properly filled in "Allow" response-header field (Section 7.4.1 of [RFC7231]).

8. The value of the HTTP "Retry-After" response-header field is taken from the value of the CoAP Max-Age Option, if present.

9. This CoAP response can only happen if the proxy itself is configured to use a CoAP forward-proxy (Section 5.7 of [RFC7252]) to execute some, or all, of its CoAP requests.

10. Only used in CoAP blockwise transfer [RFC7959] between HC Proxy and CoAP server; never translated into a HTTP response.

11. Only returned to the HTTP client if the HC Proxy was unable to successfully complete the request by retrying it with CoAP blockwise transfer; see Section 8.3.

8. Additional Mapping Guidelines

8.1. Caching and Congestion Control

An HC Proxy should cache CoAP responses, and reply whenever applicable with a cached representation of the requested resource.

If the HTTP client drops the connection after the HTTP request was made, an HC Proxy should wait for the associated CoAP response and cache it if possible. Subsequent requests to the HC Proxy for the same resource can use the result present in cache, or, if a response has still to come, the HTTP requests will wait on the open CoAP request.

According to [RFC7252], a proxy must limit the number of outstanding requests to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the HC Proxy should also put a limit on the number of concurrent CoAP requests pending on the same constrained network; further incoming requests may either be queued or dropped (returning 503 Service Unavailable). This limit and the proxy queueing/dropping behavior should be configurable.

Highly volatile resources that are being frequently requested may be observed [RFC7641] by the HC Proxy to keep their cached representation fresh while minimizing the amount of CoAP traffic in the constrained network (see Section 8.2).

8.2. Cache Refresh via Observe

There are cases where using the CoAP observe protocol [RFC7641] to handle proxy cache refresh is preferable to the validation mechanism based on ETag as defined in [RFC7252]. Such scenarios include sleepy CoAP nodes - with possibly high variance in requests' distribution - which would greatly benefit from a server-driven cache update mechanism. Ideal candidates for CoAP observe are also crowded or very low throughput networks, where reduction of the total number of exchanged messages is an important requirement.

This subsection aims at providing a practical evaluation method to decide whether refreshing a cached resource R is more efficiently handled via ETag validation or by establishing an observation on R. The idea being that the HC Proxy proactively installs an observation on a "popular enough" resource and actively monitors:

a. Its update pattern on the CoAP side; and

b. The request pattern on the HTTP side;

and uses the formula below to determine whether the observation should be kept alive or shut down.

Let T_R be the mean time between two client requests to resource R, let T_C be the mean time between two representation changes of R, and let M_R be the mean number of CoAP messages per second exchanged to and from resource R. If we assume that the initial cost for establishing the observation is negligible, an observation on R reduces M_R if and only if T_R < 2*T_C with respect to using ETag validation, that is if and only if the mean arrival rate of requests for resource R is greater than half the change rate of R.

When observing the resource R, M_R is always upper bounded by 2/T_C.

8.3. Use of CoAP Blockwise Transfer

An HC Proxy SHOULD support CoAP blockwise transfers [RFC7959] to allow transport of large CoAP payloads while avoiding excessive link-layer fragmentation in constrained networks, and to cope with small datagram buffers in CoAP endpoints as described in [RFC7252] Section 4.6.

An HC Proxy SHOULD attempt to retry a payload-carrying CoAP PUT or POST request with blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. An HC Proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than

BLOCKWISE_THRESHOLD bytes. The value of BLOCKWISE_THRESHOLD is implementation-specific; for example, it can be:

- o Calculated based on a known or typical UDP datagram buffer size for CoAP endpoints, or

- o Set to N times the known size of a link-layer frame in a constrained network where e.g., N=5, or

- o Preset to a known IP MTU value, or

- o Set to a known Path MTU value.

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

The HC Proxy SHOULD detect CoAP endpoints not supporting blockwise transfers. This can be done by checking for a 4.02 (Bad Option) response returned by an endpoint in response to a CoAP request with a Block* Option, and subsequent absence of the 4.02 in response to the same request without Block* Options. This allows the HC Proxy to be more efficient, not attempting repeated blockwise transfers to CoAP servers that do not support it.

8.4. CoAP Multicast

An HC Proxy MAY support CoAP multicast. If it does, the HC Proxy sends out a multicast CoAP request if the Target CoAP URI's authority is a multicast IP literal or resolves to a multicast IP address. If the HC Proxy does not support CoAP multicast, it SHOULD respond 403 (Forbidden) to any valid HTTP request that maps to a CoAP multicast request.

Details related to supporting CoAP multicast are currently out of scope of this document since in a proxy scenario an HTTP client typically expects to receive a single response, not multiple. However, an HC Proxy that implements CoAP multicast may include application-specific functions to aggregate multiple CoAP responses into a single HTTP response. We suggest using the "application/http" internet media type (Section 8.3.2 of [RFC7230]) to enclose a set of one or more HTTP response messages, each representing the mapping of one CoAP response.

For further considerations related to the handling of multicast requests, see Section 10.1.

8.5. Timeouts

If the CoAP server takes a long time in responding, the HTTP client or any other proxy in between may timeout. Further discussion of timeouts in HTTP is available in Section 6.2.4 of [RFC7230].

An HC Proxy MUST define an internal timeout for each pending CoAP request, because the CoAP server may silently die before completing the request. Assuming the Proxy uses confirmable CoAP requests, such timeout value T SHOULD be at least

T = MAX_RTT + MAX_SERVER_RESPONSE_DELAY

where MAX_RTT is defined in [RFC7252] and MAX_SERVER_RESPONSE_DELAY is defined in [RFC7390].

9. IANA Considerations

9.1. New 'core.hc' Resource Type

This document registers a new Resource Type (rt=) Link Target Attribute, 'core.hc', in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

Attribute Value: core.hc

Description: HTTP to CoAP mapping base resource.

Reference: See Section 5.5.

9.2. New 'coap-payload' Internet Media Type

This document defines the "application/coap-payload" media type with a single parameter "cf". This media type represents any payload that a CoAP message can carry, having a content format that can be identified by an integer in range 0-65535 corresponding to a CoAP Content-Format parameter ([RFC7252], Section 12.3). The parameter "cf" is the integer defining the CoAP content format.

Type name: application

Subtype name: coap-payload

Required parameters: cf (CoAP Content-Format integer in range 0-65535 denoting the content format of the CoAP payload carried, as defined by the "CoAP Content-Formats" subregistry that is part of the "Constrained RESTful Environments (CoRE) Parameters" registry.)

Optional parameters: None

Encoding considerations: Common use is BINARY. The specific CoAP content format encoding considerations for the selected Content-Format (cf parameter) apply. The encoding can vary based on the value of the cf parameter.

Security considerations: The specific CoAP content format security considerations for the selected Content-Format (cf parameter) apply.

Interoperability considerations: This media type can never be used directly in CoAP messages because there are no means available to encode the mandatory 'cf' parameter in CoAP.

Published specification: (this I-D - TBD)

Applications that use this media type: HTTP-to-CoAP Proxies.

Fragment identifier considerations: CoAP does not support URI fragments; therefore a CoAP payload fragment cannot be identified. Fragments are not applicable for this media type.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information:

Esko Dijk ("esko@ieee.org")

Intended usage: COMMON

Restrictions on usage:

An application (or user) can only use this media type if it has to represent a CoAP payload of which the specified CoAP Content-Format is an unrecognized number; such that a proper translation directly to the equivalent HTTP media type is not possible.

Author: CoRE WG

Change controller: IETF

Provisional registration: No

10. Security Considerations

The security considerations in Section 9.2 of [RFC7230] apply in full to the HC Proxy. This section discusses security aspects and requirements that are specific to the deployment and operation of an HC Proxy.

An HC Proxy located at the boundary of a constrained network is an easy single point of failure for reducing availability. As such, special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it has fewer limitations than the constrained devices it is serving. In particular, its quality of implementation and operation - i.e., use of current software development practices, careful selection of third party libraries, sane configuration defaults, an expedited way to upgrade a running instance - are all essential attributes of the HC Proxy.

The correctness of request parsing in general (including any content transcoding), and of URI translation in particular, is essential to the security of the HC Proxy function. This is especially true when the internal network hosts devices with genuinely limited capabilities. For this purpose, see also Sections 9.3, 9.4, 9.5 and 9.6 of [RFC7230] for well-known issues related to HTTP request parsing and Section 11.1 of [RFC7252] for an overview of CoAP specific concerns related to URI processing - in particular, the potential impact on access control mechanisms that are based on URIs.

An HC Proxy MUST implement TLS with PSK [RFC4279] and SHOULD implement TLS [RFC5246] with support for client authentication using X.509 certificates. A prerequisite of the latter is the availability of a Certification Authority (CA) to issue suitable certificates. Although this can be a challenging requirement in certain application scenarios, it is worth noting that there exist open-source tools (e.g., [OpenSSL]) that can be used to set up and operate an application-specific CA.

By default, the HC Proxy MUST authenticate all incoming requests prior to forwarding them to the CoAP server. This default behavior MAY be explicitly disabled by an administrator.

The following subparagraphs categorize and discuss a set of specific security issues related to the translation, caching and forwarding functionality exposed by an HC Proxy.

10.1. Multicast

Multicast requests impose a non-trivial cost on the constrained network and endpoints and might be exploited as a DoS attack vector (see also Section 10.2). From a privacy perspective, they can be used to gather detailed information about the resources hosted in the constrained network. For example, an outsider that is able to successfully query the /.well-known/core could obtain a comprehensive list of the target's home appliances and devices. From a security perspective, they can be used to carry out a network reconnaissance attack to gather information about possible vulnerabilities that could be exploited at a later point in time. For these reasons, it is RECOMMENDED that requests to multicast resources are access controlled with a default-deny policy. It is RECOMMENDED that the requestor of a multicast resource be strongly authenticated. If privacy and / or security are first class requirements, for example whenever the HTTP request transits through the public Internet, the request SHOULD be transported over a mutually authenticated and encrypted TLS connection.

10.2. Traffic Overflow

Due to the typically constrained nature of CoAP nodes, particular attention should be given to the implementation of traffic reduction mechanisms (see Section 8.1), because an inefficient proxy implementations can be targeted by unconstrained Internet attackers. Bandwidth or complexity involved in such attacks is very low.

An amplification attack to the constrained network may be triggered by a multicast request generated by a single HTTP request which is mapped to a CoAP multicast resource, as discussed in Section 11.3 of [RFC7252].

The risk likelihood of this amplification technique is higher than an amplification attack carried out by a malicious constrained device (e.g., ICMPv6 flooding, like Packet Too Big, or Parameter Problem on a multicast destination [RFC4732]) since it does not require direct access to the constrained network.

The feasibility of this attack, which disrupts availability of the targeted CoAP server, can be limited by access controlling the exposed multicast resources, so that only known/authorized users can access such URIs.

10.3. Handling Secured Exchanges

An HTTP request can be sent to the HC Proxy over a secured connection. However, there may not always exist a secure connection mapping to CoAP. For example, a secure distribution method for multicast traffic is complex and may not be implemented (see [RFC7390]).

An HC Proxy should implement rules for security context translations. For example, all 'https' unicast requests are translated to 'coaps' requests, or 'https' requests are translated to unsecured 'coap' requests. Another rule could specify the security policy and parameters used for DTLS sessions [RFC7925]. Such rules will largely depend on the application and network context in which the HC Proxy operates. These rules should be configurable.

It is RECOMMENDED that, by default, accessing a 'coaps' URI is only allowed from a corresponding 'https' URI.

By default, an HC Proxy SHOULD reject any secured CoAP client request (i.e., one with a 'coaps' scheme) if there is no configured security policy mapping. This recommendation may be relaxed in case the destination network is believed to be secured by other means. Assuming that CoAP nodes are isolated behind a firewall as in the HC Proxy deployment shown in Figure 1, the HC Proxy may be configured to translate the incoming HTTPS request using plain CoAP (NoSec mode).

10.4. URI Mapping

The following risks related to the URI mapping described in Section 5 and its use by HC Proxy have been identified:

DoS attack on the constrained/CoAP network.
Mitigation: by default deny any Target CoAP URI whose authority is (or maps to) a multicast address. Then explicitly white-list multicast resources/authorities that are allowed to be de-referenced. See also Section 8.4.

Leaking information on the constrained/CoAP network resources and topology.
Mitigation: by default deny any Target CoAP URI (especially /.well-known/core is a resource to be protected), and then explicitly white-list resources that are allowed to be seen from outside.

The internal CoAP Target resource is totally transparent from outside.

Mitigation: implement an HTTPS-only interface, which makes the Target CoAP URI totally opaque to a passive attacker.

11. Acknowledgments

An initial version of Table 2 in Section 7 has been provided in revision -05 of the CoRE CoAP I-D. Special thanks to Peter van der Stok for countless comments and discussions on this document that contributed to its current structure and text.

Thanks to Abhijan Bhattacharyya, Alexey Melnikov, Brian Frank, Carsten Bormann, Christian Amsuess, Christian Groves, Cullen Jennings, Dorothy Gellert, Francesco Corazza, Francis Dupont, Hannes Tschofenig, Jaime Jimenez, Kathleen Moriarty, Kepeng Li, Kerry Lynn, Klaus Hartke, Larry Masinter, Linyi Tian, Michele Rossi, Michele Zorzi, Nicola Bui, Peter Saint-Andre, Sean Leonard, Spencer Dawkins, Stephen Farrell, Suresh Krishnan, Zach Shelby for helpful comments and discussions that have shaped the document.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n.251557.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <http://www.rfc-editor.org/info/rfc4279>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <http://www.rfc-editor.org/info/rfc7232>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

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

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

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <http://www.rfc-editor.org/info/rfc7959>.

12.2. Informative References

[Fielding]
Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", PhD Dissertation, University of California, Irvine, ISBN 0-599-87118-0, 2000.

[I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-06 (work in progress), July 2016.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-09 (work in progress), October 2016.

[OpenSSL] The OpenSSL Project, , "ca - sample minimal CA application", 1998-2016, <https://www.openssl.org/docs/manmaster/man1/ca.html>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, DOI 10.17487/RFC3040, January 2001, <http://www.rfc-editor.org/info/rfc3040>.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <http://www.rfc-editor.org/info/rfc4732>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <http://www.rfc-editor.org/info/rfc6454>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <http://www.rfc-editor.org/info/rfc7390>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <http://www.rfc-editor.org/info/rfc7925>.

[W3C.REC-html5-20141028] Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, E., and S. Pfeiffer, "HTML5", W3C Recommendation REC-html5-20141028, 2014, <http://www.w3.org/TR/2014/REC-html5-20141028>.

Appendix A. Media Type Mapping Source Code

```
#!/usr/bin/env python

import unittest
import re


class CoAPContentFormatRegistry(object):
    """Map an Internet media type (and optional inherent encoding) to a
       CoAP content format.
    """
    TEXT_PLAIN = 0
    LINK_FORMAT = 40
    XML = 41
    OCTET_STREAM = 42
    EXI = 47
    JSON = 50
    CBOR = 60
    GROUP_JSON = 256

# http://www.iana.org/assignments/core-parameters/core-parameters.xhtml
# as of 2016/10/24.
    LOOKUP_TABLE = {
        ("text/plain;charset=utf-8", None): TEXT_PLAIN,
```

```
        ("application/link-format", None): LINK_FORMAT,
        ("application/xml", None): XML,
        ("application/octet-stream", None): OCTET_STREAM,
        ("application/exi", None): EXI,
        ("application/json", None): JSON,
        ("application/cbor", None): CBOR,
        ("application/coap-group+json", "utf-8"): GROUP_JSON,
    }

    def lookup(self, media_type, encoding):
        """Return the CoAP Content Format matching the supplied
           media type (and optional encoding), or None if no
           match can be found."""
        return CoAPContentFormatRegistry.LOOKUP_TABLE.get(
            (media_type, encoding), None)


class LooseMediaTypeMapper(object):
    # Order matters in this table: more specific types have higher rank
    # compared to less specific types.
    # This code only performs a shallow validation of acceptable
    # characters, and assumes overall validation of media type and
    # subtype has been done beforehand.
    LOOKUP_TABLE = [
        (re.compile("application/.+\+xml$"), "application/xml"),
        (re.compile("application/.+\+json$"), "application/json"),
        (re.compile("application/.+\+cbor$"), "application/cbor"),
        (re.compile("text/xml$"), "application/xml"),
        (re.compile("text/[a-z\.\-\+]+$"), "text/plain;charset=utf-8"),
        (re.compile("[a-z]+/[a-z\.\-\+]+$"), "application/octet-stream")
    ]

    def lookup(self, media_type):
        """Return the best loose media type match available using
           the contents of LOOKUP_TABLE."""
        for entry in LooseMediaTypeMapper.LOOKUP_TABLE:
            if entry[0].match(media_type) is not None:
                return entry[1]
        return None


def mt2cf(media_type, encoding=None,
          coap_cf_registry=CoAPContentFormatRegistry(),
          loose_mapper=None):
    """Return a CoAP Content-Format given an Internet Media Type and
       its optional encoding.  The current (as of 2016/10/24) CoAP
       Content Format Registry is supplied by default.  An optional
       'loose-mapping' implementation can be supplied by the caller."""
```

```
    assert media_type is not None
    assert coap_cf_registry is not None

    # Lookup the CoAP Content-Formats registry
    content_format = coap_cf_registry.lookup(media_type, encoding)

    # If an exact match is not found and a loose mapper has been
    # supplied, try to use it to get a media type with which to
    # re-try the CoAP Content-Formats registry lookup.
    if content_format is None and loose_mapper is not None:
        content_format = coap_cf_registry.lookup(
            loose_mapper.lookup(media_type), encoding)

    return content_format


class TestMT2CF(unittest.TestCase):

    def testMissingContentType(self):
        with self.assertRaises(AssertionError):
            mt2cf(None)

    def testMissingContentFormatRegistry(self):
        with self.assertRaises(AssertionError):
            mt2cf(None, coap_cf_registry=None)

    def testTextPlain(self):
        self.assertEqual(mt2cf("text/plain;charset=utf-8"),
                         CoAPContentFormatRegistry.TEXT_PLAIN)

    def testLinkFormat(self):
        self.assertEqual(mt2cf("application/link-format"),
                         CoAPContentFormatRegistry.LINK_FORMAT)

    def testXML(self):
        self.assertEqual(mt2cf("application/xml"),
                         CoAPContentFormatRegistry.XML)

    def testOctetStream(self):
        self.assertEqual(mt2cf("application/octet-stream"),
                         CoAPContentFormatRegistry.OCTET_STREAM)

    def testEXI(self):
        self.assertEqual(mt2cf("application/exi"),
                         CoAPContentFormatRegistry.EXI)

    def testJSON(self):
        self.assertEqual(mt2cf("application/json"),
```

```
                        CoAPContentFormatRegistry.JSON)

    def testCBOR(self):
        self.assertEqual(mt2cf("application/cbor"),
                         CoAPContentFormatRegistry.CBOR)

    def testCoAPGroupJSON(self):
        self.assertEqual(mt2cf("application/coap-group+json",
                               "utf-8"),
                         CoAPContentFormatRegistry.GROUP_JSON)

    def testUnknownMediaType(self):
        self.assertFalse(mt2cf("unknown/media-type"))

    def testLooseXML1(self):
        self.assertEqual(
            mt2cf(
                "application/somesubtype+xml",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.XML)

    def testLooseXML2(self):
        self.assertEqual(
            mt2cf(
                "text/xml",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.XML)

    def testLooseJSON(self):
        self.assertEqual(
            mt2cf(
                "application/somesubtype+json",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.JSON)

    def testLooseCBOR(self):
        self.assertEqual(
            mt2cf(
                "application/somesubtype+cbor",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.CBOR)

    def testLooseText(self):
        self.assertEqual(
            mt2cf(
                "text/somesubtype",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.TEXT_PLAIN)
```

```
    def testLooseUnknown(self):
        self.assertEqual(
            mt2cf(
                "application/somesubtype-of-some-sort+format",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.OCTET_STREAM)

    def testLooseInvalidStartsWithNonAlpha(self):
        self.assertFalse(
            mt2cf(
                " application/somesubtype",
                loose_mapper=LooseMediaTypeMapper()))

    def testLooseInvalidEndsWithUnexpectedChar(self):
        self.assertFalse(
            mt2cf(
                "application/somesubtype ",
                loose_mapper=LooseMediaTypeMapper()))

    def testLooseInvalidUnexpectedCharInTheMiddle(self):
        self.assertFalse(
            mt2cf(
                "application /somesubtype",
                loose_mapper=LooseMediaTypeMapper()))

    def testLooseInvalidNoSubType1(self):
        self.assertFalse(
            mt2cf(
                "application",
                loose_mapper=LooseMediaTypeMapper()))

    def testLooseInvalidNoSubType2(self):
        self.assertFalse(
            mt2cf(
                "application/",
                loose_mapper=LooseMediaTypeMapper()))

if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Appendix B. Change Log

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

Changes from ietf-16 to ietf-17:

- o Intended status from Informational to Standards Track;

- o Stephen Farrell's DISCUSS
- o Added 2.31 and 4.08 CoAP response codes to the Response Code Mapping table.
- o Editorial fixes

Changes from ietf-15 to ietf-16 (Apps-Dir review):

- o Larry Masinter's comments.

Changes from ietf-14 to ietf-15 (IESG review):

- o Kathleen Moriarty's DISCUSS and COMMENT;
- o Stephen Farrell's COMMENT;
- o Suresh Krishnan DISCUSS;
- o Spencer Dawkins' DISCUSS and COMMENT;

Changes from ietf-13 to ietf-14:

- o Addressed Gen-ART and AD review comments.

Changes from ietf-12 to ietf-13 (Christian Amsuess' comments):

- o More missing slashes in URI mapping template examples.

Changes from ietf-11 to ietf-12 (2nd WGLC):

- o Addressed a few editorial issues (including a clarification on when to use qq vs q in the URI mapping template).
- o Fixed missing slash in one template example.
- o Added para about the need for future CoAP protocol elements to define their own HTTP mappings.

Changes from ietf-10 to ietf-11 (Chair review):

- o Removed cu/su distinction from the URI mapping template.
- o Addressed a few editorial issues.

Changes from ietf-09 to ietf-10:

- o Addressed Ticket #401 - Clarified that draft covers not only Reverse HC Proxy but that many parts also apply to Forward and Interception Proxies.

- o Clarified that draft concentrates on the HTTP-to-CoAP mapping direction (i.e., the HC Proxy is an HTTP server and a CoAP client).

- o Clarified the "null mapping" case where no CoAP URI information is embedded in the HTTP request URI.

- o Moved multicast related security text to the "Security Considerations" to consolidate all security information in one location.

- o Removed references to "placement" of proxy (e.g., server-side vs client-side) as is confusing and provides little added value.

- o Fixed version numbers on references that were corrupted in last revision due to outdated xml2rfc conversion tool local cache.

- o Various editorial improvements.

Changes from ietf-08 to ietf-09:

- o Clean up requirements language as per Klaus' comment.

Changes from ietf-07 to ietf-08:

- o Addressed WGLC review comments from Klaus Hartke as per the correspondence of March 9, 2016 on the CORE WG mailing list.

Changes from ietf-06 to ietf-07:

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

- o Addressed Ticket #378 - For HTTP media type to CoAP content format mapping and vice versa: a new draft (TBD) may be proposed in CoRE which describes an approach for automatic updating of the media type mapping. This was noted in Section 6.1 but is otherwise outside the scope of this draft.

- o Addressed Ticket #377 - Added IANA section that defines a new HTTP media type "application/coap-payload" and created new Section 6.2 on how to use it.

- o Addressed Ticket #376 - Updated Table 2 (and corresponding note 7) to indicate that a CoAP 4.05 (Method Not Allowed) Response Code should be mapped to an HTTP 400 (Bad Request).

- o Added note to comply to ABNF when translating CoAP diagnostic payload to reason-phrase in Section 6.5.3.

Changes from ietf-05 to ietf-06:

- o Fully restructured the draft, bringing introductory text more to the front and allocating main sections to each of the key topics; addressing Ticket #379;

- o Addressed Ticket #382, fix of enhanced form URI template definition of q in Section 5.3.2;

- o Addressed Ticket #381, found a mapping 4.01 to 401 Unauthorized in Section 7;

- o Addressed Ticket #380 (Add IANA registration for "core.hc" Resource Type) in Section 9;

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

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

- o Addressed review comments of Carsten Bormann;

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

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

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

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

- o Many editorial improvements.

Changes from ietf-04 to ietf-05:

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

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

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

- o Various editorial improvements.

Changes from ietf-03 to ietf-04:

- o Expanded use case descriptions in Section 4;

- o Fixed/enhanced discovery examples in Section 5.4.1;

- o Addressed Ticket #365 (Add text on media type conversion by HTTP-CoAP proxy) in new Section 6.3.1 (Generalized media type mapping) and new Section 6.3.2 (Content translation);

- o Updated HTTPBis WG draft references to recently published RFC numbers.

- o Various editorial improvements.

Changes from ietf-02 to ietf-03:

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

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

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

Changes from ietf-01 to ietf-02:

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

Changes from ietf-00 to ietf-01:

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

Authors' Addresses

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

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal H3A 3G4
Canada

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

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

Email: thomas.fossati@nokia.com

Esko Dijk
Philips Lighting
High Tech Campus 7
Eindhoven 5656 AE
The Netherlands

Email: esko.dijk@philips.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 30, 2018

K. Li
Alibaba Group
A. Rahman
InterDigital
C. Bormann, Ed.
Universitaet Bremen TZI
February 26, 2018

Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR

draft-ietf-core-links-json-10

Abstract

JavaScript Object Notation, JSON (RFC 8259) is a text-based data format which is popular for Web based data exchange. Concise Binary Object Representation, CBOR (RFC7049) is a binary data format which has been optimized for data exchange for the Internet of Things (IoT). For many IoT scenarios, CBOR formats will be preferred since it can help decrease transmission payload sizes as well as implementation code sizes compared to other data formats.

Web Linking (RFC 8288) provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format (RFC 6690). Outside of constrained environments, it may be useful to represent these collections of Web links in JSON, and similarly, inside constrained environments, in CBOR. This specification defines a common format for this.

Status of This Memo

Table of Contents

1. Introduction

Web Linking [RFC8288] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252].

The JavaScript Object Notation (JSON) [RFC8259] is a lightweight, text-based, language-independent data interchange format. JSON is popular in the Web development environment as it is easy for humans to read and write.

The Concise Binary Object Representation (CBOR) [RFC7049] is a binary data format which requires extremely small code size, allows very compact message representation, and provides extensibility without the need for version negotiation. CBOR is especially well suited for IoT environments because of these efficiencies.

When converting between a bespoke syntax such as that defined by [RFC6690] and JSON or CBOR, many small decisions have to be made. If left without guidance, it is likely that a number of slightly incompatible dialects will emerge. This specification defines a common format for representing CoRE Web Linking in JSON and CBOR.

Note that there is a separate question on how to represent Web links pointing out of JSON documents, as discussed for example in [MNOT11]. While there are good reasons to stay as compatible as possible to developments in this area, the present specification is solving a different problem.

1.1. Objectives

This specification has been designed based on the following objectives:

- o Canonical mapping
  - * lossless conversion in both directions between any pair of [RFC6690], JSON, and CBOR ("round-tripping"), unless prevented by a limitation of [RFC6690]
  - * but not attempting to ensure that a sequence of conversions from one of the formats through one or both of the others and back to the original would result in a bit-wise identical representation
- o The simplest thing that could possibly work.

While the formats defined in this document are based on the above objectives, they are general enough that they can be used for other applications of links in the Web. The same basic formats can be used for Web links that do not default to the "hosts" relation type (as is defined in [RFC6690]) and that allow percent encoding and general IRI syntax in what is an URI-Reference field in [RFC6690]. Also, specific support has been added for internationalized link attributes

such as "title*", including their language tags (while staying limited to UTF-8 as the character set).

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

CoAP: Constrained Application Protocol [RFC7252]

CBOR: Concise Binary Object Representation [RFC7049]

CoRE: Constrained RESTful Environments, the field of work underlying [RFC6690], [RFC7049], [RFC7252], [RFC7641], [RFC7959], [RFC8075], and [RFC8323]

IoT: Internet of Things

JSON: JavaScript Object Notation [RFC8259]

The objective of the JSON and CBOR mappings defined in this document is to contain information of the formats specified in [RFC8288] and [RFC6690]. This specification therefore uses the names of the ABNF productions used in those documents.

2. Web Links in JSON and CBOR

2.1. Background

Web Linking [RFC8288] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252] and in conjunction with the CoRE resource directory [I-D.ietf-core-resource-directory].

2.2. Information Model

This section discusses the information model underlying the CORE Link Format payload.

An "application/link-format" document is a collection of Web links ("link-value"), each of which is a collection of attributes ("link-param") applied to a "URI-Reference".

We straightforwardly map:

- o the collection of Web links to a JSON or CBOR array of links;

- o each link to a JSON object or CBOR map, mapping attribute names to attribute values.

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a JSON name/value pair (member). The name is a string representation of the parameter or attribute name (as in "parmname"). The value can be a string, a language-tagged string, a boolean, or an array of these, as described below.

If the attribute value ("ptoken" or "quoted-string") is present, and a Link attribute with this name ("parmname") is present just once in the "link-value", the value is a string representation of the parameter or attribute value ("ptoken" or "quoted-string"). "quoted-string" productions are parsed (i.e, the outer quotes removed and the backslash constructions evaluated) as defined in [RFC6690] and its referenced documents, before placing them in JSON strings (in the representation of which they may gain back additional decorations such as backslashes as defined in [RFC8259]).

Attribute values represented as per [RFC8187], e.g. for the "title*" attribute, are converted in a language-tagged string; the attribute name is then represented without the "*" character. A language-tagged string is represented as a CBOR map (JSON object) that carries the language tag as the key for a single member and the attribute value in UTF-8 form as its value.

If no attribute value ("ptoken" or "quoted-string") is present, the presence of the attribute name is indicated by using the Boolean value "true" as the value.

If a Link attribute ("parmname") is present more than once in a "link-value", its values are then represented as a JSON array of JSON string values or "true"; this array becomes the value of the JSON name/value pair where the attribute name is the JSON name. Attributes occurring just once MUST NOT be represented as JSON arrays but MUST be directly represented as JSON strings or "true". (Note that [RFC6690] has cut down on the use of repeated parameter names; they are still allowed by [RFC8288] though. No attempt has been made to decode the possibly space-separated values for rt=, if=, and rel=

into JSON arrays.) Recipients MUST NOT accept documents that violate this requirement.

The URI-Reference is represented as a name/value pair with the name "href" and the URI-Reference as the value, with the latter converted to an IRI-Reference as per Section 3.2 of [RFC3987] (Rationale: The usage of "href" is consistent with the use of "href" as a query parameter for link-format query filtering and with link-format reserving the link parameter "href" specifically for this use [RFC6690]. The usage of an IRI-Reference is consistent with the mandate in [RFC6690] that percent-encoding be processed. Note that the format is able to represent IRIs the URIs for which cannot be represented in [RFC6690] as not all percent-encoded constructions are amenable to the pre-processing required by [RFC6690].)

As a convenient reference, the resulting structure can be described in CBOR Data Definition Language (CDDL) [I-D.ietf-cbor-cddl] as in Figure 1 (informative).

```
links = [* link]
link = {
  href: tstr    ; resource URI
  * tstr => value
}
value1 = tstr   ; text value -- the normal case
       / { tstr => tstr } ; language tag and value
       / true   ; no value given, just the name
value = value1
      / [2* value1 ] ; repeats for two or more
```

Figure 1: CoRE Link Format Data Model (JSON)

2.3. Additional Encoding Step for CBOR

The above specification for JSON might have been used as is for the CBOR encoding as well. However, to further reduce message sizes, an extra encoding step is performed: "href" and some commonly occurring attribute names are encoded as small integers.

The substitution is defined in Table 1:

name	encoded value	origin
href	1	[RFC6690], [RFCthis]
rel	2	[RFC5988] Section 5.3
anchor	3	[RFC5988] Section 5.2
rev	4	[RFC5988] Section 5.3
hreflang	5	[RFC5988] Section 5.4
media	6	[RFC5988] Section 5.4
title	7	[RFC5988] Section 5.4
type	8	[RFC5988] Section 5.4
rt	9	[RFC6690] Section 3.1
if	10	[RFC6690] Section 3.2
sz	11	[RFC6690] Section 3.3
ct	12	[RFC7252] Section 7.2.1
obs	13	[RFC7641] Section 6

Table 1: Integer Encoding of common attribute names

This list of substitutions is fixed by the present specification; no future expansion of the list is foreseen. "href" as well as all attribute names in this list MUST be represented by their integer substitutions and MUST NOT use the attribute name in text form. Recipients MUST NOT accept documents that violate this requirement.

As a convenient reference, the resulting structure can be described in CBOR Data Definition Language (CDDL) [I-D.ietf-cbor-cddl] as in Figure 2 (informative).

```
links = [* link]
link = {
  href => tstr    ; resource URI
  * label => value
}
href = 1
label = tstr / &(
  rel: 2,       anchor: 3,  rev: 4,
  hreflang: 5,  media: 6,   title: 7,
  type: 8,      rt: 9,      if: 10,
  sz: 11,       ct: 12,     obs: 13,
)
value1 = tstr   ; text value -- the normal case
       / { tstr => tstr } ; language tag and value
       / true   ; no value given, just the name
value = value1
      / [2* value1 ] ; repeats for two or more
```

Figure 2: CoRE Link Format Data Model (CBOR)

2.4. Converting JSON or CBOR to Link-Format

When a JSON or CBOR representation needs to be converted back to link-format, the above process is performed in inverse. Since link-format allows serializing link parameter values both in unqouted form ("ptoken") or in quoted form ("quoted-string"), a decision has to be made for each value. Where the syntax of "ptoken" does not allow the value to be represented, the quoted form clearly needs to be used. However, when both forms are possible, the decision is arbitrary. The recently republished Web Linking specification, [RFC8288], clarifies that this is indeed intended to be the case. However, previous specifications of link attributes, including those in [RFC5988] and [RFC6690], sometimes have made this decision in a specific way by only including one or the other alternative in the ABNF given for a link parameter. This requires a converter to know about all these cases, including those that have not been defined yet at the time of writing the converter. This problem becomes even harder by the fact that there is no central registry of link-attribute names.

Obviously, the conversion back to link-format needs to result in a valid link-format document. The reference implementation in Appendix A has addressed this problem with the following two rules:

- Where a "ptoken" representation is possible, that is used instead of "quoted-string". This rule covers most of the special cases listed above.

- o As a special exception to the above rule, the four link attributes "anchor", "title", "rt", and "if" are always expressed as "quoted-string". This rule covers these specific four cases.

This set of rules is based on the hope that future definitions of link attributes will no longer hardcode one or the other serialization.

2.5. Examples

The examples in this section are based on an example on page 15 of [RFC6690] (Figure 3).

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 3: Example from page 15 of [RFC6690]

2.5.1. Link Format to JSON Example

The link-format document in Figure 3 becomes (321 bytes, line breaks shown are not part of the minimally-sized JSON document):

```
"[{"href":"/sensors","ct":"40","title":"Sensor
Index"},{"href":"/sensors/temp","rt":"temperature-
c","if":"sensor"},{"href":"/sensors/light","rt":"light-
lux","if":"sensor"},{"href":"http://www.example.com/sensors/
t123","anchor":"/sensors/
temp","rel":"describedby"},{"href":"/t","anchor":"/sensors/
temp","rel":"alternate"}] "
```

To demonstrate the handling of value-less and array-valued attributes, we extend the link-format example by examples of these (Figure 4; the "obs" attribute is defined in Section 6 of [RFC7641], while the "foo" attribute is for exposition only):

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor";obs,
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby";foo="bar";foo=3;ct=4711,
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 4: Example derived from page 15 of [RFC6690]

The link-format document in Figure 4 becomes the JSON document in Figure 5 (some spacing and indentation added):

```
[{"href":"/sensors","ct":"40","title":"Sensor Index"},
 {"href":"/sensors/temp","rt":"temperature-c","if":"sensor",
  "obs":true},
 {"href":"/sensors/light","rt":"light-lux","if":"sensor"},
 {"href":"http://www.example.com/sensors/t123",
  "anchor":"/sensors/temp","rel":"describedby",
  "foo":["bar","3"],"ct":"4711"},
 {"href":"/t","anchor":"/sensors/temp","rel":"alternate"}]
```

Figure 5: Example derived from page 15 of [RFC6690]

Note that the conversion is unable to convert the string-valued "ct" attribute to a number, which would be the natural type for a Content-Format value; similarly, both "foo" values are treated as strings independently of whether they are quoted or numeric in syntax.

2.5.2. Link Format to CBOR Example

This examples shows conversion from link format to CBOR format.

The link-format document in Figure 3 becomes (in CBOR diagnostic format):

```
[{1: "/sensors", 12: "40", 7: "Sensor Index"},
 {1: "/sensors/temp", 9: "temperature-c", 10: "sensor"},
 {1: "/sensors/light", 9: "light-lux", 10: "sensor"},
 {1: "http://www.example.com/sensors/t123", 3: "/sensors/temp",
  2: "describedby"},
 {1: "/t", 3: "/sensors/temp", 2: "alternate"}]
```

or, in hexadecimal (203 bytes):

```
85                                # array(number of data items:5)
   a3                             # map(# data item pairs:3)
      01                          # unsigned integer(value:1,"href")
      68                          # text string(8 bytes)
         2f73656e736f7273         # "/sensors"
      0c                          # unsigned integer(value:12,"ct")
      62                          # text(2)
         3430                     # "40"
      07                          # unsigned integer(value:7,"title")
      6c                          # text string(12 bytes)
         53656e736f7220496e646578 # "Sensor Index"
   a3                             # map(# data item pairs:3)
      01                          # unsigned integer(value:1,"href")
```

```
         6d                          # text string(13 bytes)
            2f73656e736f72732f74
            656d70                   # "/sensors/temp"
         09                          # unsigned integer(value:9,"rt")
         6d                          # text string(13 bytes)
            74656d70657261747572
            652d63                   # "temperature-c"
         0a                          # unsigned integer(value:10,"if")
         66                          # text string(6 bytes)
            73656e736f72              # "sensor"
      a3                             # map(# data item pairs:3)
         01                          # unsigned integer(value:1,"href")
         6e                          # text string(14 bytes)
            2f73656e736f72732f6c
            69676874                 # "/sensors/light"
         09                          # unsigned integer(value:9,"rt")
         69                          # text string(9 bytes)
            6c696768742d6c7578       # "light-lux"
         0a                          # unsigned integer(value:10,"if")
         66                          # text string(6 bytes)
            73656e736f72              # "sensor"
      a3                             # map(# data item pairs:3)
         01                          # unsigned integer(value:1,"href")
         78 23                       # text string(35 bytes)
            687474703a2f2f777777
            2e6578616d706c652e63
            6f6d2f73656e736f7273
            2f74313233           # "http://www.example.com/sensors/t123"
         03                          # unsigned integer(value:3,"anchor")
         6d                          # text string(13 bytes)
            2f73656e736f72732f74
            656d70                   # "/sensors/temp"
         02                          # unsigned integer(value:2,"rel")
         6b                          # text string(11 bytes)
            6465736372696265646279   # "describedby"
      a3                             # map(# data item pairs:3)
         01                          # unsigned integer(value:1,"href")
         62                          # text string(2 bytes)
            2f74                     # "/t"
         03                          # unsigned integer(value:3,"anchor")
         6d                          # text string(13 bytes)
            2f73656e736f72732f74
            656d70                   # "/sensors/temp"
         02                          # unsigned integer(value:2,"rel")
         69                          # text string(9 bytes)
            616c7465726e617465       # "alternate"
```

Figure 6: Web Links Encoded in CBOR

3. IANA Considerations

3.1. Media types

This specification registers the following additional Internet Media Types:

Type name: application

Subtype name: link-format+json

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+json" media type are required to conform to the "application/json" Media Type and are therefore subject to the same encoding considerations specified in [RFC8259], Section 11.

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

Applications that use this media type: Applications that interchange collections of Web links based on CoRE link format [RFC6690] in JSON.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information: Carsten Bormann <cabo@tzi.org>

Intended usage: COMMON

Change controller: IESG

and

Type name: application

Subtype name: link-format+cbor

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+cbor" media type are required to conform to the "application/cbor" Media Type and are therefore subject to the same encoding considerations specified in [RFC7049], Section 7.

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

Applications that use this media type: Applications that interchange collections of Web links based on CoRE link format [RFC6690] in CBOR.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): CBOR

Person & email address to contact for further information: Kepeng Li <kepeng.lkp@alibaba-inc.com>

Intended usage: COMMON

Change controller: IESG

3.2. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the above media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. The ID for "application/link-format+cbor" is assigned from the "Expert Review" (0-255) range, while the ID for "application/link-format+json" is assigned from the "IETF review" range. The assigned IDs are show in Table 2.

Media type	Coding	ID	Reference
application/link-format+cbor	-	TBD64	[RFCthis]
application/link-format+json	-	TBD504	[RFCthis]

Table 2: CoAP Content-Format IDs

4. Security Considerations

The security considerations relevant to the data model of [RFC6690], as well as those of [RFC7049] and [RFC8259] apply.

5. References

5.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8187] Reschke, J., "Indicating Character Encoding and Language for HTTP Header Field Parameters", RFC 8187, DOI 10.17487/RFC8187, September 2017, <https://www.rfc-editor.org/info/rfc8187>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

5.2. Informative References

[I-D.ietf-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-01 (work in progress), January 2018.

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-12 (work in progress), October 2017.

[MNOT11] Nottingham, M., "Linking in JSON", November 2011, <http://www.mnot.net/blog/2011/11/25/linking_in_json>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <https://www.rfc-editor.org/info/rfc5988>.

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

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

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

[RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <https://www.rfc-editor.org/info/rfc8075>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

[RUBY] "Information technology -- Programming languages -- Ruby", ISO/IEC 30170:2012, April 2012.

Appendix A. Reference implementation

A reference implementation of a converter from [RFC6690] link-format to JSON and CBOR (and back to link-format) in the programming language Ruby [RUBY] is reproduced below. (Note that this implementation does not handle [RFC8187]-encoded attributes.) For pretty-printing the binary CBOR, this uses the "cbor-diag" gem (Ruby library), which may need to be installed by "gem install cbor-diag".

```
# <CODE BEGINS>
require 'strscan'
require 'json'
require 'cbor-pretty'

class String
  def as_utf8
    force_encoding(Encoding::UTF_8)
  end
end

module CoRE
  module Links
    def self.map_to_true(a)
      Hash[a.map{ |t| [t, true]}]
    end

    PTOKENCHAR = %r"[\[\]\w!#-+\--/:<-?^-`{-~@]"
    QUOSTRCHAR = %r{(?:[^"\\]|\\.)}    # to be used inside "
    ATTRCHAR   = %r"[\w!#$&+.^`|~-]"
    MUSTBEQUOTED = map_to_true(%w{anchor title rt if})
    ANCHORNAME = "href"
    SCANATTR =
%r{(#{ATTRCHAR}+)(?:=(?:(#{PTOKENCHAR}+)|"(#{QUOSTRCHAR}*)"))?} # "

    RAWMAPPINGS = <<-DATA
  href: 1,   rel: 2,        anchor: 3,
  rev: 4,    hreflang: 5,   media: 6,
  title: 7,  type: 8,       rt: 9,
  if: 10,    sz: 11,        ct: 12,
```

```
     obs: 13,
       DATA

       MAPPINGS = Hash.new {|h, k| k}

       RAWMAPPINGS.scan(/([-\w]+)\s*:\s*([-\w]+),/) do |n, v|
         MAPPINGS[n] = Integer(v)
       end

       def self.parse(*args)
         WLNK.parse(*args)
       end

       class WLNK
         attr_accessor :resources
         def initialize(r = [])    # make sure the keys are strings
           @resources = r.to_ary   # make sure it's an Array
         end
         def self.parse(s, robust = true)
           wl = WLNK.new
           ss = StringScanner.new(s.as_utf8)
           ss.skip(/\s+/) if robust
           while ss.scan(%r{<([^>]+)>})
             res = { ANCHORNAME => ss[1].as_utf8 }
             ss.skip(/\s*/) if robust
             while ss.skip(/;/)
               ss.skip(/\s*/) if robust
               unless ss.scan(SCANATTR)
                 raise ArgumentError, "must have attribute behind ';'
                   at: #{ss.peek(20).inspect} (byte #{ss.pos})"
               end
               key = ss[1].as_utf8
               value = ss[2] ||
                       (ss[3] ? ss[3].gsub(/\\(.)/) { $1 } : true)
               if res[key]
                 res[key] = Array(res[key]) << value
               else
                 res[key] = value
               end
               ss.skip(/\s*/) if robust
             end
             wl.resources << res
             break unless ss.skip(/,/)
             ss.skip(/\s*/) if robust
           end
           ss.skip(/\s*/) if robust
           raise ArgumentError, "link-format unparseable at:
              #{ss.peek(20).inspect} (byte #{ss.pos})" unless ss.eos?
```

```
          wl
        end
        def to_json
          JSON.pretty_generate(@resources)
        end
        def to_cbor
          CBOR.encode(@resources.map {|r|
                        Hash[r.map { |k, v| [MAPPINGS[k], v] }]})
        end
        def to_wlnk
          resources.map do |res|
            res = res.dup
            u = res.delete(ANCHORNAME)
            ["<#{u}>", *res.map { |k, v| wlnk_item(k, v) }].join(';')
          end.join(",")
        end
        private
        def wlnk_item(k, v)
          case v
          when String
            if MUSTBEQUOTED[k] || v !~ /\A#{PTOKENCHAR}+\z/
              "#{k}=\"#{v.gsub(/[\\"]/) { |x| "\\#{x}"}}\""
            else
              "#{k}=#{v}"
            end
          when Array
            v.map{ |v1| wlnk_item(k, v1) }.join(';')
          when true
            "#{k}"
          else
            fail "Don't know how to represent #{{k=>v}.inspect}"
          end
        end
      end
    end
  end

  lf = CoRE::Links.parse(ARGF.read)

  puts lf.to_json                 # JSON
  puts CBOR.pretty(lf.to_cbor)    # CBOR "pretty" binary form
  puts lf.to_wlnk                 # RFC 6690 link-format
  # <CODE ENDS>
```

Acknowledgements

Special thanks to Bert Greevenbosch who was an author on the initial version of a contributing document as well as the original author on the CDDL notation.

Hannes Tschofenig made many helpful suggestions for improving this document.

Authors' Addresses

Kepeng LI
Alibaba Group
Wenyixi Road, Yuhang District
Hangzhou, Zhejiang 311121
China

Email: kepeng.lkp@alibaba-inc.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal, Quebec H3A 3G4
Canada

Phone: +1-514-585-0761
Email: akbar.rahman@interdigital.com

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

K. Hartke
Universitaet Bremen TZI

October 27, 2014

Observing Resources in CoAP
draft-ietf-core-observe-15

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

Copyright Notice

Table of Contents

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.

```
                      Observer             Subject
                         |                    |
                         |    Registration    |
                         +------------------->|
                         |                    |
                         |    Notification    |
                         |<-------------------+
                         |                    |
                         |    Notification    |
                         |<-------------------+
                         |                    |
                         |    Notification    |
                         |<-------------------+
                         |                    |
```

Figure 1: The Observer Design Pattern

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.

```
                        Client                Server
                           |                    |
                           |  GET /temperature  |
                           |    Token: 0x4a     |   Registration
                           |  Observe: register |
                           +------------------->|
                           |                    |
                           |    2.05 Content    |
                           |    Token: 0x4a     |   Notification of
                           |  Observe: 12       |   the current state
                           |  Payload: 22.9 Cel |
                           |<-------------------+
                           |                    |
                           |    2.05 Content    |
                           |    Token: 0x4a     |   Notification upon
                           |  Observe: 44       |   a state change
                           |  Payload: 22.8 Cel |
                           |<-------------------+
                           |                    |
                           |    2.05 Content    |
                           |    Token: 0x4a     |   Notification upon
                           |  Observe: 60       |   a state change
                           |  Payload: 23.1 Cel |
                           |<-------------------+
                           |                    |
```

Figure 2: Observing a Resource in CoAP

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;

- o <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

No.	C	U	N	R	Name	Format	Length	Default
6		x	-		Observe	uint	0-3 B	(none)

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 and V2 - V1 < 2^23) or
(V1 > V2 and V1 - V2 > 2^23) or
(T2 > T1 + 128 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 observed 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 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:

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

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

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

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

10.2. Informative References

[GOF] Gamma, E., Helm, R., Johnson, R., and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, USA, November 1994.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[RFC5989] Roach, A., "A SIP Event Package for Subscribing to Changes to an HTTP Resource", RFC 5989, October 2010.

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.

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

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

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

Appendix A. Examples

A.1. Client/Server Examples

```
         Observed   CLIENT  SERVER     Actual
     t   State          |       |         State
         _____________  |       |  _____________
     1                  |       |
     2    unknown       |       |     18.5 Cel
     3                 +----->|                   Header: GET 0x41011633
     4                 | GET  |                    Token: 0x4a
     5                 |       |                Uri-Path: temperature
     6                 |       |                 Observe: 0 (register)
     7                 |       |
     8                 |       |
     9   _____________  <-----+                   Header: 2.05 0x61451633
    10                 | 2.05 |                    Token: 0x4a
    11    18.5 Cel     |       |                 Observe: 9
    12                 |       |                 Max-Age: 15
    13                 |       |                 Payload: "18.5 Cel"
    14                 |       |
    15                 |       |  _____________
    16   _____________  <-----+                   Header: 2.05 0x51457b50
    17                 | 2.05 |     19.2 Cel       Token: 0x4a
    18    19.2 Cel     |       |                 Observe: 16
    29                 |       |                 Max-Age: 15
    20                 |       |                 Payload: "19.2 Cel"
    21                 |       |
```

Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change

```
         Observed   CLIENT  SERVER     Actual
     t   State          |       |         State
         _____________  |       |  _____________
    22                  |       |
    23    19.2 Cel      |       |     19.2 Cel
    24                  |       |  _____________
    25                  | X----+                  Header: 2.05 0x51457b51
    26                  | 2.05 |     19.7 Cel      Token: 0x4a
    27                  |       |                 Observe: 25
    28                  |       |                 Max-Age: 15
    29                  |       |                 Payload: "19.7 Cel"
    30                  |       |
    31   _____________  |       |
    32                  |       |
    33    19.2 Cel      |       |
    34    (stale)       |       |
    35                  |       |
    36                  |       |
    37                  |       |
    38                 +----->|                  Header: GET 0x41011634
    39                  | GET  |                  Token: 0xb2
    40                  |       |               Uri-Path: temperature
    41                  |       |                 Observe: 0 (register)
    42                  |       |
    43                  |       |
    44   _____________  |<-----+                  Header: 2.05 0x61451634
    45                  | 2.05 |                  Token: 0xb2
    46    19.7 Cel      |       |                 Observe: 44
    47                  |       |                 Max-Age: 15
    48                  |       |                    ETag: 0x78797a7a79
    49                  |       |                 Payload: "19.7 Cel"
    50                  |       |
```

Figure 4: The client re-registers after Max-Age ends

```
         Observed   CLIENT  SERVER     Actual
    t    State          |       |         State
         _____________  |       |  _____________
   51                   |       |
   52     19.7 Cel      |       |      19.7 Cel
   53                   |       |
   54                   |       |  _____________
   55                   |     crash
   56                   |
   57                   |
   58                   |
   59    _____________  |
   60                   |
   61     19.7 Cel      |
   62     (stale)       |
   63                   |    reboot_____________
   64                   |       |
   65                   |       |      20.0 Cel
   66                   |       |
   67                   +----->|                   Header: GET 0x41011635
   68                   | GET   |                    Token: 0xf9
   69                   |       |                Uri-Path: temperature
   70                   |       |                 Observe: 0 (register)
   71                   |       |                    ETag: 0x78797a7a79
   72                   |       |
   73                   |       |
   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"
   79                   |       |
   80                   |       |  _____________
   81    _____________  |<-----+                   Header: 2.03 0x5143aa0c
   82                   | 2.03  |      19.7 Cel      Token: 0xf9
   83     19.7 Cel      |       |                 Observe: 81
   84                   |       |                    ETag: 0x78797a7a79
   85                   |       |                 Max-Age: 15
   86                   |       |
```

Figure 5: The client re-registers and gives the server the opportunity to select a stored response

```
         Observed   CLIENT  SERVER     Actual
    t    State          |       |         State
         _____________  |       |  _____________
   87                   |       |
   88     19.7 Cel      |       |      19.7 Cel
   89                   |       |
   90                   |       |  _____________
   91    _____________  |<------+                 Header: 2.05 0x4145aa0f
   92                   | 2.05  |      19.3 Cel    Token: 0xf9
   93     19.3 Cel      |       |                Observe: 91
   94                   |       |                Max-Age: 15
   95                   |       |                Payload: "19.3 Cel"
   96                   |       |
   97                   |       |
   98                   +- - ->|                 Header: 0x7000aa0f
   99                   |       |
  100                   |       |
  101                   |       |
  102                   |       |  _____________
  103                   |       |
  104                   |       |      19.0 Cel
  105                   |       |
  106    _____________  |       |
  107                   |       |
  108     19.3 Cel      |       |
  109     (stale)       |       |
  110                   |       |
```

Figure 6: The client rejects a notification and thereby cancels the observation

A.2. Proxy Examples

```
   CLIENT  PROXY  SERVER
      |       |       |
      |       +------>|     Header: GET 0x41015fb8
      |       | GET   |      Token: 0x1a
      |       |       |  Uri-Host: sensor.example
      |       |       |  Uri-Path: status
      |       |       |   Observe: 0 (register)
      |       |       |
      |       |<------+     Header: 2.05 0x61455fb8
      |       | 2.05  |      Token: 0x1a
      |       |       |   Observe: 42
      |       |       |   Max-Age: 60
      |       |       |   Payload: "ready"
      |       |       |
      +------>|       |     Header: GET 0x41011633
      | GET   |       |      Token: 0x9a
      |       |       | Proxy-Uri: coap://sensor.example/status
      |       |       |
      |<------+       |     Header: 2.05 0x61451633
      | 2.05  |       |      Token: 0x9a
      |       |       |   Max-Age: 53
      |       |       |   Payload: "ready"
      |       |       |
      |       |<------+     Header: 2.05 0x514505fc0
      |       | 2.05  |      Token: 0x1a
      |       |       |   Observe: 135
      |       |       |   Max-Age: 60
      |       |       |   Payload: "busy"
      |       |       |
      +------>|       |     Header: GET 0x41011634
      | GET   |       |      Token: 0x9b
      |       |       | Proxy-Uri: coap://sensor.example/status
      |       |       |
      |<------+       |     Header: 2.05 0x61451634
      | 2.05  |       |      Token: 0x9b
      |       |       |   Max-Age: 49
      |       |       |   Payload: "busy"
      |       |       |
```

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

```
   CLIENT  PROXY  SERVER
      |       |       |
      +----->|       |     Header: GET 0x41011635
      | GET   |       |      Token: 0x6a
      |       |       | Proxy-Uri: coap://sensor.example/status
      |       |       |   Observe: 0 (register)
      |       |       |
      |<- - -+        |     Header: 0x60001635
      |       |       |
      |       +----->|     Header: GET 0x4101af90
      |       | GET   |      Token: 0xaa
      |       |       |  Uri-Host: sensor.example
      |       |       |  Uri-Path: status
      |       |       |   Observe: 0 (register)
      |       |       |
      |       |<-----+     Header: 2.05 0x6145af90
      |       | 2.05  |      Token: 0xaa
      |       |       |   Observe: 67
      |       |       |   Max-Age: 60
      |       |       |   Payload: "ready"
      |       |       |
      |<-----+        |     Header: 2.05 0x4145af94
      | 2.05  |       |      Token: 0x6a
      |       |       |   Observe: 17346
      |       |       |   Max-Age: 60
      |       |       |   Payload: "ready"
      |       |       |
      +- - ->|        |     Header: 0x6000af94
      |       |       |
      |       |<-----+     Header: 2.05 0x51455a20
      |       | 2.05  |      Token: 0xaa
      |       |       |   Observe: 157
      |       |       |   Max-Age: 60
      |       |       |   Payload: "busy"
      |       |       |
      |<-----+        |     Header: 2.05 0x5145af9b
      | 2.05  |       |      Token: 0x6a
      |       |       |   Observe: 17436
      |       |       |   Max-Age: 60
      |       |       |   Payload: "busy"
      |       |       |
```

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-14 to ietf-15:

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

Changes from ietf-13 to ietf-14:

- o Updated references.

Changes from ietf-12 to ietf-13:

- o 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'.

- o Removed the 7.31 Code for cancellation.

Changes from ietf-11 to ietf-12:

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

- o Made the algorithm for superseding an outstanding transmission OPTIONAL.

- o 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).

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

Changes from ietf-10 to ietf-11:

- o 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:

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

- o 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.

- o 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.

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

- o Updated examples to use UCUM units.

- o Moved Appendix B into the introduction.

Changes from ietf-08 to ietf-09:

- o 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).

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

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

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

Changes from ietf-07 to ietf-08:

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

- o Changed reordering detection to use a fixed time span of 128 seconds instead of EXCHANGE_LIFETIME (#276).

- o 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.

- o 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.

- o 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.

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

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

- o 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.

- o 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.

- o 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.

- 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.
- Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

- 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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 3, 2015

K. Hartke
Universitaet Bremen TZI

December 30, 2014

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

Copyright Notice

Table of Contents

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.

```
                      Observer             Subject
                         |                    |
                         |    Registration    |
                         +------------------->|
                         |                    |
                         |    Notification    |
                         |<-------------------+
                         |                    |
                         |    Notification    |
                         |<-------------------+
                         |                    |
                         |    Notification    |
                         |<-------------------+
                         |                    |
```

Figure 1: The Observer Design Pattern

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.

```
                        Client                Server
                           |                    |
                           |  GET /temperature  |
                           |    Token: 0x4a     |   Registration
                           |  Observe: 0        |
                           +------------------->|
                           |                    |
                           |    2.05 Content    |
                           |    Token: 0x4a     |   Notification of
                           |  Observe: 12       |   the current state
                           |  Payload: 22.9 Cel |
                           |<-------------------+
                           |                    |
                           |    2.05 Content    |
                           |    Token: 0x4a     |   Notification upon
                           |  Observe: 44       |   a state change
                           |  Payload: 22.8 Cel |
                           |<-------------------+
                           |                    |
                           |    2.05 Content    |
                           |    Token: 0x4a     |   Notification upon
                           |  Observe: 60       |   a state change
                           |  Payload: 23.1 Cel |
                           |<-------------------+
                           |                    |
```

Figure 2: Observing a Resource in CoAP

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

exceeds the threshold, or to "OK" when the reading drops below;

- o <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

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

No.	C	U	N	R	Name	Format	Length	Default
6		x	-		Observe	uint	0-3 B	(none)

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 and V2 - V1 < 2^23) or
(V1 > V2 and V1 - V2 > 2^23) or
(T2 > T1 + 128 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, 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 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 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 observed 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 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:

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

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

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

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

10.2. Informative References

[GOF] Gamma, E., Helm, R., Johnson, R., and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, USA, November 1994.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[RFC5989] Roach, A., "A SIP Event Package for Subscribing to Changes to an HTTP Resource", RFC 5989, October 2010.

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.

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

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

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

Appendix A. Examples

A.1. Client/Server Examples

```
         Observed   CLIENT  SERVER     Actual
     t   State          |       |         State
         _____________  |       |  _____________
     1                  |       |
     2    unknown       |       |     18.5 Cel
     3                 +----->|                   Header: GET 0x41011633
     4                 | GET  |                    Token: 0x4a
     5                 |       |                Uri-Path: temperature
     6                 |       |                 Observe: 0 (register)
     7                 |       |
     8                 |       |
     9   _____________  <-----+                   Header: 2.05 0x61451633
    10                 | 2.05 |                    Token: 0x4a
    11    18.5 Cel     |       |                 Observe: 9
    12                 |       |                 Max-Age: 15
    13                 |       |                 Payload: "18.5 Cel"
    14                 |       |
    15                 |       |  _____________
    16   _____________  <-----+                   Header: 2.05 0x51457b50
    17                 | 2.05 |     19.2 Cel       Token: 0x4a
    18    19.2 Cel     |       |                 Observe: 16
    29                 |       |                 Max-Age: 15
    20                 |       |                 Payload: "19.2 Cel"
    21                 |       |
```

Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change

```
         Observed   CLIENT  SERVER     Actual
    t    State          |       |         State
         _____________  |       |  _____________
   22                   |       |
   23    19.2 Cel       |       |     19.2 Cel
   24                   |       |  _____________
   25                   | X----+                  Header: 2.05 0x51457b51
   26                   | 2.05 |     19.7 Cel      Token: 0x4a
   27                   |       |                 Observe: 25
   28                   |       |                 Max-Age: 15
   29                   |       |                 Payload: "19.7 Cel"
   30                   |       |
   31   _____________   |       |
   32                   |       |
   33    19.2 Cel       |       |
   34    (stale)        |       |
   35                   |       |
   36                   |       |
   37                   |       |
   38                  +----->|                  Header: GET 0x41011634
   39                   | GET  |                  Token: 0xb2
   40                   |       |               Uri-Path: temperature
   41                   |       |                 Observe: 0 (register)
   42                   |       |
   43                   |       |
   44   _____________   |<-----+                  Header: 2.05 0x61451634
   45                   | 2.05 |                  Token: 0xb2
   46    19.7 Cel       |       |                 Observe: 44
   47                   |       |                 Max-Age: 15
   48                   |       |                    ETag: 0x78797a7a79
   49                   |       |                 Payload: "19.7 Cel"
   50                   |       |
```

Figure 4: The client re-registers after Max-Age ends

```
         Observed   CLIENT  SERVER     Actual
     t   State          |       |         State
         _____________  |       |  _____________
    51                  |       |
    52    19.7 Cel      |       |      19.7 Cel
    53                  |       |
    54                  |       |  _____________
    55                  |     crash
    56                  |
    57                  |
    58                  |
    59   _____________  |
    60                  |
    61    19.7 Cel      |
    62    (stale)       |
    63                  |    reboot_____________
    64                  |       |
    65                  |       |      20.0 Cel
    66                  |       |
    67                  +----->|                   Header: GET 0x41011635
    68                  | GET   |                    Token: 0xf9
    69                  |       |                Uri-Path: temperature
    70                  |       |                 Observe: 0 (register)
    71                  |       |                    ETag: 0x78797a7a79
    72                  |       |
    73                  |       |
    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"
    79                  |       |
    80                  |       |  _____________
    81   _____________  |<-----+                   Header: 2.03 0x5143aa0c
    82                  | 2.03  |      19.7 Cel      Token: 0xf9
    83    19.7 Cel      |       |                 Observe: 81
    84                  |       |                    ETag: 0x78797a7a79
    85                  |       |                 Max-Age: 15
    86                  |       |
```

Figure 5: The client re-registers and gives the server the opportunity to select a stored response

```
         Observed   CLIENT  SERVER     Actual
    t    State          |       |         State
         _____________  |       |  _____________
   87                   |       |
   88     19.7 Cel      |       |      19.7 Cel
   89                   |       |
   90                   |       |  _____________
   91    _____________  |<------+                 Header: 2.05 0x4145aa0f
   92                   | 2.05  |      19.3 Cel    Token: 0xf9
   93     19.3 Cel      |       |                Observe: 91
   94                   |       |                Max-Age: 15
   95                   |       |                Payload: "19.3 Cel"
   96                   |       |
   97                   |       |
   98                   +- - ->|                 Header: 0x7000aa0f
   99                   |       |
  100                   |       |
  101                   |       |
  102                   |       |  _____________
  103                   |       |
  104                   |       |      19.0 Cel
  105                   |       |
  106    _____________  |       |
  107                   |       |
  108     19.3 Cel      |       |
  109     (stale)       |       |
  110                   |       |
```

Figure 6: The client rejects a notification and thereby cancels the observation

A.2. Proxy Examples

```
   CLIENT  PROXY  SERVER
      |       |       |
      |       +------>|     Header: GET 0x41015fb8
      |       | GET   |      Token: 0x1a
      |       |       |  Uri-Host: sensor.example
      |       |       |  Uri-Path: status
      |       |       |   Observe: 0 (register)
      |       |       |
      |       |<------+     Header: 2.05 0x61455fb8
      |       | 2.05  |      Token: 0x1a
      |       |       |   Observe: 42
      |       |       |   Max-Age: 60
      |       |       |   Payload: "ready"
      |       |       |
      +------>|       |     Header: GET 0x41011633
      | GET   |       |      Token: 0x9a
      |       |       | Proxy-Uri: coap://sensor.example/status
      |       |       |
      |<------+       |     Header: 2.05 0x61451633
      | 2.05  |       |      Token: 0x9a
      |       |       |   Max-Age: 53
      |       |       |   Payload: "ready"
      |       |       |
      |       |<------+     Header: 2.05 0x514505fc0
      |       | 2.05  |      Token: 0x1a
      |       |       |   Observe: 135
      |       |       |   Max-Age: 60
      |       |       |   Payload: "busy"
      |       |       |
      +------>|       |     Header: GET 0x41011634
      | GET   |       |      Token: 0x9b
      |       |       | Proxy-Uri: coap://sensor.example/status
      |       |       |
      |<------+       |     Header: 2.05 0x61451634
      | 2.05  |       |      Token: 0x9b
      |       |       |   Max-Age: 49
      |       |       |   Payload: "busy"
      |       |       |
```

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

```
   CLIENT  PROXY  SERVER
      |       |       |
      +------>|       |     Header: GET 0x41011635
      | GET   |       |      Token: 0x6a
      |       |       | Proxy-Uri: coap://sensor.example/status
      |       |       |   Observe: 0 (register)
      |       |       |
      |<- - -+        |     Header: 0x60001635
      |       |       |
      |       +------>|     Header: GET 0x4101af90
      |       | GET   |      Token: 0xaa
      |       |       |  Uri-Host: sensor.example
      |       |       |  Uri-Path: status
      |       |       |   Observe: 0 (register)
      |       |       |
      |       |<------+     Header: 2.05 0x6145af90
      |       | 2.05  |      Token: 0xaa
      |       |       |   Observe: 67
      |       |       |   Max-Age: 60
      |       |       |   Payload: "ready"
      |       |       |
      |<------+       |     Header: 2.05 0x4145af94
      | 2.05  |       |      Token: 0x6a
      |       |       |   Observe: 17346
      |       |       |   Max-Age: 60
      |       |       |   Payload: "ready"
      |       |       |
      +- - ->|        |     Header: 0x6000af94
      |       |       |
      |       |<------+     Header: 2.05 0x51455a20
      |       | 2.05  |      Token: 0xaa
      |       |       |   Observe: 157
      |       |       |   Max-Age: 60
      |       |       |   Payload: "busy"
      |       |       |
      |<------+       |     Header: 2.05 0x5145af9b
      | 2.05  |       |      Token: 0x6a
      |       |       |   Observe: 17436
      |       |       |   Max-Age: 60
      |       |       |   Payload: "busy"
      |       |       |
```

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:

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

Changes from ietf-14 to ietf-15:

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

Changes from ietf-13 to ietf-14:

- o Updated references.

Changes from ietf-12 to ietf-13:

- o 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'.

- o Removed the 7.31 Code for cancellation.

Changes from ietf-11 to ietf-12:

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

- o Made the algorithm for superseding an outstanding transmission OPTIONAL.

- o 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).

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

Changes from ietf-10 to ietf-11:

- o 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:

- o Required consistent sequence numbers across requests (#333).
- o 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.
- o 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.
- o Improved text on the transmission of non-confirmable notifications to match Section 3.1.2 of RFC 5405 more closely.
- o Updated examples to use UCUM units.
- o Moved Appendix B into the introduction.

Changes from ietf-08 to ietf-09:

- o 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).
- o Clarified that a resource (and hence an observation relationship) is identified by the request options that are part of the Cache-Key (#258).
- o Clarified that a non-2.xx notification MUST NOT include an Observe Option.
- o Moved block-wise transfer of notifications to [I-D.ietf-core-block].

Changes from ietf-07 to ietf-08:

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

- o Changed reordering detection to use a fixed time span of 128 seconds instead of EXCHANGE_LIFETIME (#276).

- o 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.

- o 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.

- o 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.

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

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

- o 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.

- o 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.

- o 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.

- Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

- 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

CoRE
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2021

C. Amsüss, Ed.

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

7 March 2021

CoRE Resource Directory
draft-ietf-core-resource-directory-28

Abstract

In many IoT applications, direct discovery of resources is not practical due to sleeping nodes, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources. The input to an RD is composed of links and the output is composed of links constructed from the information stored in the RD. This document specifies the web interfaces that an RD supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with an RD are defined.

Note to Readers

Discussion of this document takes place on the CORE Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/ (https://mailarchive.ietf.org/arch/browse/core/).

Source for this draft and an issue tracker can be found at https://github.com/core-wg/resource-directory (https://github.com/core-wg/resource-directory).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In the work on Constrained RESTful Environments (CoRE), a REST architecture suitable for constrained nodes (e.g. with limited RAM and ROM [RFC7228]) and networks (e.g. 6LoWPAN [RFC4944]) has been established and is used in Internet-of-Things (IoT) or machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC8288]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. However, [RFC6690] only describes how to discover resources from the web server that hosts them by querying "/.well-known/core". In many constrained scenarios, direct discovery of resources is not practical due to sleeping nodes, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that an RD supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with an RD are defined. Although the examples in this document show the use of these interfaces with CoAP [RFC7252], they can be applied in an equivalent manner to HTTP [RFC7230].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC3986], [RFC8288] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

resolve against
: The expression "a URI-reference is _resolved against_ a base URI" is used to describe the process of [RFC3986] Section 5.2. Noteworthy corner cases are that if the URI-reference is a (full) URI and resolved against any base URI, that gives the original full URI, and that resolving an empty URI reference gives the base URI without any fragment identifier.

Resource Directory (RD)
: A web entity that stores information about web resources and implements the REST interfaces defined in this specification for discovery, for the creation, maintenance and removal of registrations, and for lookup of the registered resources.

Sector
: In the context of an RD, a sector is a logical grouping of endpoints.

  The abbreviation "d=" is used for the sector in query parameters for compatibility with deployed implementations.

Endpoint
: Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the RD. An endpoint is identified by its endpoint name, which is included during registration, and has a unique name within the associated sector of the registration.

Registration Base URI
: The Base URI of a Registration is a URI that typically gives scheme and authority information about an Endpoint. The Registration Base URI is provided at registration time, and is used by the RD to resolve relative references of the registration into URIs.

Target
The target of a link is the destination address (URI) of the link. It is sometimes identified with "href=", or displayed as "<target>". Relative targets need resolving with respect to the Base URI (section 5.2 of [RFC3986]).

This use of the term Target is consistent with [RFC8288]'s use of the term.

Context
The context of a link is the source address (URI) of the link, and describes which resource is linked to the target. A link's context is made explicit in serialized links as the "anchor=" attribute.

This use of the term Context is consistent with [RFC8288]'s use of the term.

Directory Resource
A resource in the RD containing registration resources.

Registration Resource
A resource in the RD that contains information about an Endpoint and its links.

Commissioning Tool
Commissioning Tool (CT) is a device that assists during installation events by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

Registrant-ep
Registrant-ep is the endpoint that is registered into the RD. The registrant-ep can register itself, or a CT registers the registrant-ep.

RDAO
Resource Directory Address Option. A new IPv6 Neighbor Discovery option defined for announcing an RD's address.

3. Architecture and Use Cases

3.1. Principles

The RD is primarily a tool to make discovery operations more efficient than querying /.well-known/core on all connected devices, or across boundaries that would limit those operations.

It provides information about resources hosted by other devices that could otherwise only be obtained by directly querying the /.well-known/core resource on these other devices, either by a unicast request or a multicast request.

Information SHOULD only be stored in the RD if it can be obtained by querying the described device's /.well-known/core resource directly.

Data in the RD can only be provided by the device which hosts those data or a dedicated Commissioning Tool (CT). These CTs act on behalf of endpoints too constrained, or generally unable, to present that information themselves. No other client can modify data in the RD. Changes to the information in the RD do not propagate automatically back to the web servers from where the information originated.

3.2. Architecture

The RD architecture is illustrated in Figure 1. An RD is used as a repository of registrations describing resources hosted on other web servers, also called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port. A physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain RD registrations, and for endpoints to lookup resources from the RD. An RD can be logically segmented by the use of Sectors.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Registrations in the RD are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a registration. It is also possible for an RD to fetch Web Links from endpoints and add their contents to its registrations.

At the first registration of an endpoint, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registrations.

A lookup interface for discovering any of the Web Links stored in the RD is provided using the CoRE Link Format.

```
                Registration        Lookup
                 Interface         Interface
     +----+          |                 |
     | EP |----      |                 |
     +----+    ----  |                 |
                   --|-    +------+    |
     +----+          | ----|      |    |     +--------+
     | EP | ---------|-----|  RD  |----|-----| Client |
     +----+          | ----|      |    |     +--------+
                   --|-    +------+    |
     +----+    ----  |                 |
     | CT |----      |                 |
     +----+
```

Figure 1: The RD architecture.

A Registrant-EP MAY keep concurrent registrations to more than one RD at the same time if explicitly configured to do so, but that is not expected to be supported by typical EP implementations. Any such registrations are independent of each other. The usual expectation when multiple discovery mechanisms or addresses are configured is that they constitute a fall-back path for a single registration.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 2 and Figure 3 model the contents of /.well-known/core and the RD respectively, with entity-relationship diagrams [ER]. Entities (rectangles) are used for concepts that exist independently. Attributes (ovals) are used for concepts that exist only in connection with a related entity. Relations (diamonds) give a semantic meaning to the relation between entities. Numbers specify the cardinality of the relations.

Some of the attribute values are URIs. Those values are always full URIs and never relative references in the information model. They can, however, be expressed as relative references in serializations, and often are.

These models provide an abstract view of the information expressed in link-format documents and an RD. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

```
                      +----------------------+
                      |   /.well-known/core  |
                      +----------------------+
                                 |
                                 | 1
                        ////////\\\\\\\
                       <    contains   >
                        \\\\\\\\///////
                                 |
                                 | 0+
                       +--------------------+
                       |      link          |
                       +--------------------+
                                 |
                                 |  1   oooooooo
                                +-----o target o
                                 |      oooooooo
            oooooooooooo   0+    |
           o    target  o--------+
           o  attribute o        | 0+   oooooo
            oooooooooooo         +-----o rel  o
                                 |      oooooo
                                 |
                                 | 1    ooooooooo
                                +-----o context o
                                       ooooooooo
```

Figure 2: ER Model of the content of /.well-known/core

The model shown in Figure 2 models the contents of /.well-known/core which contains:

* a set of links belonging to the hosting web server

The web server is free to choose links it deems appropriate to be exposed in its "/.well-known/core". Typically, the links describe resources that are served by the host, but the set can also contain links to resources on other servers (see examples in [RFC6690] page 14). The set does not necessarily contain links to all resources served by the host.

A link has the following attributes (see [RFC8288]):

* Zero or more link relations: They describe relations between the link context and the link target.

  In link-format serialization, they are expressed as space-separated values in the "rel" attribute, and default to "hosts".

* A link context URI: It defines the source of the relation, e.g. _who_ "hosts" something.

  In link-format serialization, it is expressed in the "anchor" attribute and defaults to the Origin of the target (practically: the target with its path and later components removed)

* A link target URI: It defines the destination of the relation (e.g. _what_ is hosted), and is the topic of all target attributes.

  In link-format serialization, it is expressed between angular brackets, and sometimes called the "href".

* Other target attributes (e.g. resource type (rt), interface (if), or content format (ct)). These provide additional information about the target URI.

```
                    +--------------+
                    +      RD      +
                    +--------------+
                            | 1
                            |
                            |
                            |
                      //////\\\\
                     < contains >
                      \\\\\//////
                            |
                         0+ |
    ooooooo     1  +---------------+
   o  base o-------|  registration |
    ooooooo        +---------------+
                        |       | 1
                        |       +--------------+
         ooooooooo   1 |                      |
        o  href  o----+                 /////\\\\
         ooooooooo     |                < contains >
                       |                 \\\\\//////
         ooooooooo   1 |                      |
        o   ep   o----+                      | 0+
         ooooooooo     |             +------------------+
                       |             |      link        |
         ooooooooo 0-1 |             +------------------+
        o    d   o----+                      |
         ooooooooo     |                      |  1   ooooooooo
                       |                      +-----o target o
         ooooooooo   1 |                      |      ooooooooo
        o   lt   o----+     oooooooooooo   0+ |
         ooooooooo     |    o  target   o-----+
                       |    o attribute o     | 0+   oooooo
      ooooooooooo 0+   |     oooooooooooo     +-----o rel  o
     o  endpoint o----+                      |      oooooo
     o attribute o                           |
      ooooooooooo                            | 1   oooooooooo
                                             +----o context o
                                                  oooooooooo
```

Figure 3: ER Model of the content of the RD

The model shown in Figure 3 models the contents of the RD which contains in addition to /.well-known/core:

* 0 to n Registrations of endpoints,

A registration is associated with one endpoint. A registration defines a set of links as defined for /.well-known/core. A Registration has six types of attributes:

* an endpoint name ("ep", a Unicode string) unique within a sector

* a Registration Base URI ("base", a URI typically describing the scheme://authority part)

* a lifetime ("lt"),

* a registration resource location inside the RD ("href"),

* optionally a sector ("d", a Unicode string)

* optional additional endpoint attributes (from Section 9.3)

The cardinality of "base" is currently 1; future documents are invited to extend the RD specification to support multiple values (e.g. [I-D.silverajan-core-coap-protocol-negotiation]). Its value is used as a Base URI when resolving URIs in the links contained in the endpoint.

Links are modelled as they are in Figure 2.

3.4. Link-local addresses and zone identifiers

Registration Base URIs can contain link-local IP addresses. To be usable across hosts, those cannot be serialized to contain zone identifiers (see [RFC6874] Section 1).

Link-local addresses can only be used on a single link (therefore RD servers cannot announce them when queried on a different link), and lookup clients using them need to keep track of which interface they got them from.

Therefore, it is advisable in many scenarios to use addresses with larger scope if available.

3.5. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded wireless interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. The ambition in such systems is to build them from reusable components. These speed up

development and deployment, and enable shared use of machines across different applications. One crucial component of such systems is the discovery of resources (and thus the endpoints they are hosted on) capable of providing required information at a given time or acting on instructions from the end users.

Imagine a scenario where endpoints installed on vehicles enable tracking of the position of these vehicles for fleet management purposes and allow monitoring of environment parameters. During the boot-up process endpoints register with an RD, which is hosted by the mobile operator or somewhere in the cloud. Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web applications for environment monitoring contact the RD, look up the endpoints capable of providing information about the environment using an appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server), and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look up for EPs deployed on the vehicles the application is responsible for.

3.6. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of IoT web services. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices. Both can use the common RD infrastructure to establish device interactions efficiently, but can pick security policies suitable for their needs.

Two phases can be discerned for a network servicing the system: (1) installation and (2) operation. During the operational phase, the network is connected to the Internet with a Border Router (e.g. a 6LoWPAN Border Router (6LBR), see [RFC6775]) and the nodes connected to the network can use the Internet services that are provided by the Internet Provider or the network administrator. During the installation phase, the network is completely stand-alone, no Border Router is connected, and the network only supports the IP communication between the connected nodes. The installation phase is

usually followed by the operational phase. As an RD's operations work without hard dependencies on names or addresses, it can be used for discovery across both phases.

3.7. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. An RD can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to an RD. Applications wishing to consume the data can use RD Lookup to discover and resolve links to the desired resources and endpoints. The RD service need not be coupled with the data intermediary service. Mapping of RDs to data intermediaries may be many-to-many.

Metadata in web link formats like [RFC6690] which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links, need to be supported by RDs. External catalogues that are represented in other formats may be converted to common web linking formats for storage and access by RDs. Since it is common practice for these to be encoded in URNs [RFC8141], simple and lossless structural transforms should generally be sufficient to store external metadata in RDs.

The additional features of an RD allow sectors to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Application groups with multicast addresses may be defined to support efficient data transport.

4. RD discovery and other interface-independent components

This and the following sections define the required set of REST interfaces between an RD, endpoints and lookup clients. Although the examples throughout these sections assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. The multicast discovery and simple registration operations are exceptions to that, as they rely on mechanisms unavailable in HTTP. In all definitions in these sections, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces.

All operations on the contents of the RD MUST be atomic and idempotent.

For several operations, interface templates are given in list form; those describe the operation participants, request codes, URIs, content formats and outcomes. Sections of those templates contain normative content about Interaction, Method, URI Template and URI Template Variables as well as the details of the Success condition. The additional sections on options like Content-Format and on Failure codes give typical cases that an implementation of the RD should deal with. Those serve to illustrate the typical responses to readers who are not yet familiar with all the details of CoAP based interfaces; they do not limit what a server may respond under atypical circumstances.

REST clients (registrant-EPs and CTs during registration and maintenance, lookup clients, RD servers during simple registrations) must be prepared to receive any unsuccessful code and act upon it according to its definition, options and/or payload to the best of their capabilities, falling back to failing the operation if recovery is not possible. In particular, they SHOULD retry the request upon 5.03 (Service Unavailable; 503 in HTTP) according to the Max-Age (Retry-After in HTTP) option, and SHOULD fall back to link-format when receiving 4.15 (Unsupported Content-Format; 415 in HTTP).

An RD MAY make the information submitted to it available to further directories (subject to security policies on link confidentiality), if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

4.1. Finding a Resource Directory

A (re-)starting device may want to find one or more RDs before it can discover their URIs. Dependent on the operational conditions, one or more of the techniques below apply.

The device may be pre-configured to exercise specific mechanisms for finding the RD:

1. It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an appropriate RD. (Instead of using an anycast address, a multicast address can also be preconfigured. The RD servers then need to configure one of their interfaces with this multicast address.)

2. It may be configured with a DNS name for the RD and use DNS to return the IP address of the RD; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.

3. It may be configured to use a service discovery mechanism such as DNS-SD, as outlined in Section 4.1.2.

For cases where the device is not specifically configured with a way to find an RD, the network may want to provide a suitable default.

1. The IPv6 Neighbor Discovery option RDAO Section 4.1.1 can do that.

2. When DHCP is in use, this could be provided via a DHCP option (no such option is defined at the time of writing).

Finally, if neither the device nor the network offers any specific configuration, the device may want to employ heuristics to find a suitable RD.

The present specification does not fully define these heuristics, but suggests a number of candidates:

1. In a 6LoWPAN, just assume the Border Router (6LBR) can act as an RD (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".

2. In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[MCD1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

When answering a multicast request directed at a link-local group, the RD may want to respond from a routable address; this makes it easier for registrants to use one of their own routable addresses for

registration. When [RFC6724] is used for source address selection, this can be achieved by applying the changes of its Section 10.4, picking public addresses in its Section 5 Rule 7, and superseding rule 8 with preferring the source address's precedence.

As some of the RD addresses obtained by the methods listed here are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

The following RD discovery mechanisms are recommended:

* In managed networks with border routers that need stand-alone operation, the RDAO option is recommended (e.g. operational phase described in Section 3.6).

* In managed networks without border router (no Internet services available), the use of a preconfigured anycast address is recommended (e.g. installation phase described in Section 3.6).

* In networks managed using DNS-SD, the use of DNS-SD for discovery as described in Section 4.1.2 is recommended.

The use of multicast discovery in mesh networks is NOT RECOMMENDED.

4.1.1. Resource Directory Address Option (RDAO)

The Resource Directory Address Option (RDAO) carries information about the address of the RD in RAs (Router Advertisements) of IPv6 Neighbor Discovery (ND), similar to how RDNSS options [RFC8106] are sent. This information is needed when endpoints cannot discover the RD with a link-local or realm-local scope multicast address, for instance because the endpoint and the RD are separated by a Border Router (6LBR). In many circumstances the availability of DHCP cannot be guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many RD addresses.

The RDAO format is:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |  Length = 3   |           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Valid Lifetime                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                          RD Address                           +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

Type: TBD38

Length: 8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3.

Reserved: This field is unused. It MUST be initialized to zero by the sender and MUST be ignored by the receiver.

Valid Lifetime: 32-bit unsigned integer. The length of time in seconds (relative to the time the packet is received) that this RD address is valid. A value of all zero bits (0x0) indicates that this RD address is not valid anymore.

RD Address: IPv6 address of the RD.

Figure 4: Resource Directory Address Option

4.1.2. Using DNS-SD to discover a Resource Directory

An RD can advertise its presence in DNS-SD [RFC6763] using the service name "_core-rd._udp" (for CoAP), "_core-rd-dtls._udp" (for CoAP over DTLS), "_core-rd._tcp" (for CoAP over TCP) or "_core-rd-tls._tcp" (for CoAP over TLS) defined in this document. (For the WebSocket transports of CoAP, no service is defined as DNS-SD is typically unavailable in environments where CoAP over WebSockets is used).

The selection of the service indicates the protocol used, and the SRV record points the client to a host name and port to use as a starting point for the URI discovery steps of Section 4.3.

This section is a simplified concrete application of the more generic mechanism specified in [I-D.ietf-core-rd-dns-sd].

4.2. Payload Content Formats

RDs implementing this specification MUST support the application/link-format content format (ct=40).

RDs implementing this specification MAY support additional content formats.

Any additional content format supported by an RD implementing this specification SHOULD be able to express all the information expressible in link-format. It MAY be able to express information that is inexpressible in link-format, but those expressions SHOULD be avoided where possible.

4.3. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690] after having discovered a host as described in Section 4.1.

Discovery of the RD registration URI is performed by sending either a multicast or unicast GET request to "/.well-known/core" and including a Resource Type (rt) parameter [RFC6690] with the value "core.rd" in the query string. Likewise, a Resource Type parameter value of "core.rd-lookup*" is used to discover the URIs for RD Lookup operations, core.rd* is used to discover all URIs for RD operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI of the RD function returned and the corresponding Resource Type. When

performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network (see Section 9.5).

An RD MAY provide hints about the content-formats it supports in the links it exposes or registers, using the "ct" target attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the RD.

HTTP does not support multicast and consequently only unicast discovery can be supported at the using the HTTP "/.well-known/core" resource.

RDs implementing this specification MUST support query filtering for the rt parameter as defined in [RFC6690].

While the link targets in this discovery step are often expressed in path-absolute form, this is not a requirement. Clients of the RD SHOULD therefore accept URIs of all schemes they support, both as URIs and relative references, and not limit the set of discovered URIs to those hosted at the address used for URI discovery.

With security policies where the client requires the RD to be authorized to act as an RD, that authorization may be limited to resources on which the authorized RD advertises the adequate resource types. Clients that have obtained links they can not rely on yet can repeat the URI discovery step at the /.well-known/core resource of the indicated host to obtain the resource type information from an authorized source.

The URI Discovery operation can yield multiple URIs of a given resource type. The client of the RD can use any of the discovered addresses initially.

The discovery request interface is specified as follows (this is exactly the Well-Known Interface of [RFC6690] Section 4, with the additional requirement that the server MUST support query filtering):

Interaction: EP, CT or Client -> RD

Method: GET

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

URI Template Variables: rt := Resource Type. SHOULD contain one of the values "core.rd", "core.rd-lookup*", "core.rd-lookup-res", "core.rd-lookup-ep", or "core.rd*"

Accept: absent, application/link-format or any other media type representing web links

The following response is expected on this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format or other web link payload containing one or more matching entries for the RD resource.

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource location, in this example, is /rd, and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that it is up to the RD to choose its RD locations.

```
Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
Payload:
</rd>;rt=core.rd;ct=40,
</rd-lookup/ep>;rt=core.rd-lookup-ep;ct=40,
</rd-lookup/res>;rt=core.rd-lookup-res;ct=40
```

Figure 5: Example discovery exchange

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as a CBOR and JSON representation from [I-D.ietf-core-links-json] (which have no numeric values assigned yet, so they are shown as TBD64 and TBD504 as in that draft). The RD resource locations /rd, and /rd-lookup are example values. The server in this example also indicates that it is capable of providing observation on resource lookups.

```
Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
Payload:
</rd>;rt=core.rd;ct="40 65225",
</rd-lookup/res>;rt=core.rd-lookup-res;ct="40 TBD64 TBD504";obs,
</rd-lookup/ep>;rt=core.rd-lookup-ep;ct="40 TBD64 TBD504"
```

Figure 6: Example discovery exchange indicating additional content-formats

For maintenance, management and debugging, it can be useful to identify the components that constitute the RD server. The identification can be used to find client-server incompatibilities, supported features, required updates and other aspects. The Well-Known interface described in Section 4 of [RFC6690] can be used to find such data.

It would typically be stored in an implementation information link (as described in [I-D.bormann-t2trg-rel-impl]):

```
Req: GET /.well-known/core?rel=impl-info

Res: 2.05 Content
Payload:
<http://software.example.com/shiny-resource-directory/1.0beta1>;
    rel=impl-info
```

Figure 7: Example exchange of obtaining implementation information, using the relation type currently proposed in the work-in-progress document

Note that depending on the particular server's architecture, such a link could be anchored at the RD server's root (as in this example), or at individual RD components. The latter is to be expected when different applications are run on the same server.

5. Registration

After discovering the location of an RD, a registrant-ep or CT MAY register the resources of the registrant-ep using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690] or other representations of web links, along with query parameters indicating the name of the endpoint, and optionally the sector, lifetime and base URI of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. The receiving endpoint MUST use that location when refreshing registrations using this interface. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The creating endpoint is responsible for refreshing the registration resource within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d (sector) does not create multiple registration resources.

The following rules apply for a registration request targeting a given (ep, d) value pair:

* When the (ep, d) value pair of the registration-request is different from any existing registration, a new registration is generated.

* When the (ep, d) value pair of the registration-request is equal to an existing registration, the content and parameters of the existing registration are replaced with the content of the registration request. Like the later changes to registration resources, security policies (Section 7) usually require such requests to come from the same device.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. They are resolved against the base URI of the registration, which is provided either explicitly in the "base" parameter or constructed implicitly from the requester's URI as constructed from its network address and scheme.

For media types to which Appendix C applies (i.e. documents in application/link-format), request bodies MUST be expressed in Limited Link Format.

The registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+rd}{?ep,d,lt,base,extra-attrs*}

URI Template Variables: rd := RD registration URI (mandatory).
This is the location of the RD, as obtained from discovery.

ep := Endpoint name (mostly mandatory).
The endpoint name is an identifier that MUST be unique within a sector.

As the endpoint name is a Unicode string, it is encoded in UTF-8 (and possibly pct-encoded) during variable expansion (see [RFC6570] Section 3.2.1). The endpoint name MUST NOT contain any character in the inclusive ranges 0-31 or 127-159.

The maximum length of this parameter is 63 UTF-8 encoded bytes.

If the RD is configured to recognize the endpoint to be authorized to use exactly one endpoint name, the RD assigns that name. In that case, giving the endpoint name becomes optional for the client; if the client gives any other endpoint name, it is not authorized to perform the registration.

d := Sector (optional). The sector to which this endpoint belongs. When this parameter is not present, the RD MAY associate the endpoint with a configured default sector (possibly based on the endpoint's authorization) or leave it empty.

The sector is encoded like the ep parameter, and is limited to 63 UTF-8 encoded bytes as well.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included in the initial registration, a default value of 90000 (25 hours) SHOULD be assumed.

base := Base URI (optional). This parameter sets the base URI of the registration, under which the relative links in the payload are to be interpreted. The specified URI typically does not have a path component of its own, and MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore usually of the shape "scheme://authority" for HTTP and CoAP URIs. The URI SHOULD NOT have a query or fragment component as any non-empty relative part in a reference would remove those parts from the resulting URI.

In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. The Base URI is consecutively constructed by concatenating the used protocol's scheme with the characters "://", the requester's source address as an address literal and ":" followed by its port (if it was not the protocol's default one) in analogy to [RFC7252] Section 6.5.

This parameter is mandatory when the directory is filled by a third party such as an commissioning tool.

If the registrant-ep uses an ephemeral port to register with, it MUST include the base parameter in the registration to provide a valid network path.

A registrant that cannot be reached by potential lookup clients

at the address it registers from (e.g. because it is behind some form of Network Address Translation (NAT)) MUST provide a reachable base address with its registration.

If the Base URI contains a link-local IP literal, it MUST NOT contain a Zone Identifier, and MUST be local to the link on which the registration request is received.

Endpoints that register with a base that contains a path component cannot efficiently express their registrations in Limited Link Format (Appendix C). Those applications should use different representations of links to which Appendix C is not applicable (e.g. [I-D.hartke-t2trg-coral]).

extra-attrs := Additional registration attributes (optional). The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

Content-Format: application/link-format or any other indicated media type representing web links

The following response is expected on this interface:

Success: 2.01 "Created" or 201 "Created". The Location-Path option or Location header field MUST be included in the response. This location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links.

A registration with an already registered ep and d value pair responds with the same success code and location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

The location MUST NOT have a query or fragment component, as that could conflict with query parameters during the Registration Update operation. Therefore, the Location-Query option MUST NOT be present in a successful response.

If the registration fails, including request timeouts, or if delays from Service Unavailable responses with Max-Age or Retry-After accumulate to exceed the registrant's configured timeouts, it SHOULD

pick another registration URI from the "URI Discovery" step and if there is only one or the list is exhausted, pick other choices from the "Finding a Resource Directory" step. Care has to be taken to consider the freshness of results obtained earlier, e.g. of the result of a "/.well-known/core" response, the lifetime of an RDAO option and of DNS responses. Any rate limits and persistent errors from the "Finding a Resource Directory" step must be considered for the whole registration time, not only for a single operation.

The following example shows a registrant-ep with the name "node1" registering two resources to an RD using this interface. The location "/rd" is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel=describedby

Res: 2.01 Created
Location-Path: /rd/4521
```

Figure 8: Example registration payload

An RD may optionally support HTTP. Here is an example of almost the same registration operation above, when done using HTTP.

```
Req:
POST /rd?ep=node1&base=http://[2001:db8:1::1] HTTP/1.1
Host: rd.example.com
Content-Type: application/link-format

</sensors/temp>;rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel=describedby

Res:
HTTP/1.1 201 Created
Location: /rd/4521
```

Figure 9: Example registration payload as expressed using HTTP

5.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to an RD as described in Section 5. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification MUST implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the registrant-ep makes available the hosted resources that it wants to be discovered, as links on its "/.well-known/core" interface as specified in [RFC6690]. The links in that document are subject to the same limitations as the payload of a registration (with respect to Appendix C).

* The registrant-ep finds one or more addresses of the directory server as described in Section 4.1.

* The registrant-ep sends (and regularly refreshes with) a POST request to the "/.well-known/rd" URI of the directory server of choice. The body of the POST request is empty, and triggers the resource directory server to perform GET requests at the requesting registrant-ep's /.well-known/core to obtain the link-format payload to register.

  The registrant-ep includes the same registration parameters in the POST request as it would with a regular registration per Section 5. The registration base URI of the registration is taken from the registrant-ep's network address (as is default with regular registrations).

  Example request from registrant-EP to RD (unanswered until the next step):

```
Req: POST /.well-known/rd?lt=6000&ep=node1
(No payload)
```

  Figure 10: First half example exchange of a simple registration

* The RD queries the registrant-ep's discovery resource to determine the success of the operation. It SHOULD keep a cache of the discovery resource and not query it again as long as it is fresh.

  Example request from the RD to the registrant-EP:

```
Req: GET /.well-known/core
Accept: 40

Res: 2.05 Content
Content-Format: 40
Payload:
</sen/temp>
```

Figure 11: Example exchange of the RD querying the simple endpoint

With this response, the RD would answer the previous step's request:

```
Res: 2.04 Changed
```

Figure 12: Second half example exchange of a simple registration

The sequence of fetching the registration content before sending a successful response was chosen to make responses reliable, and the point about caching was chosen to still allow very constrained registrants. Registrants MUST be able to serve a GET request to "/.well-known/core" after having requested registration. Constrained devices MAY regard the initial request as temporarily failed when they need RAM occupied by their own request to serve the RD's GET, and retry later when the RD already has a cached representation of their discovery resources. Then, the RD can reply immediately and the registrant can receive the response.

The simple registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: /.well-known/rd{?ep,d,lt,extra-attrs*}

URI Template Variables are as they are for registration in Section 5. The base attribute is not accepted to keep the registration interface simple; that rules out registration over CoAP-over-TCP or HTTP that would need to specify one. For some time during this document's development, the URI template "/.well-known/core{?ep,...}" has been in use instead.

The following response is expected on this interface:

Success: 2.04 "Changed".

For the second interaction triggered by the above, the registrant-ep takes the role of server and the RD the role of client. (Note that this is exactly the Well-Known Interface of [RFC6690] Section 4):

Interaction: RD -> EP

Method: GET

URI Template: /.well-known/core

The following response is expected on this interface:

Success: 2.05 "Content".

When the RD uses any authorization credentials to access the endpoint's discovery resource, or when it is deployed in a location where third parties might reach it but not the endpoint, it SHOULD verify that the apparent registrant-ep intends to register with the given registration parameters before revealing the obtained discovery information to lookup clients. An easy way to do that is to verify the simple registration request's sender address using the Echo option as described in [I-D.ietf-core-echo-request-tag] Section 2.4.

The RD MUST delete registrations created by simple registration after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

5.2. Third-party registration

For some applications, even Simple Registration may be too taxing for some very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the RD can be filled by a third party device, called a Commissioning Tool (CT). The commissioning tool can fill the RD from a database or other means. For that purpose scheme, IP address and port of the URI of the registered device is the value of the "base" parameter of the registration described in Section 5.

It should be noted that the value of the "base" parameter applies to all the links of the registration and has consequences for the anchor value of the individual links as exemplified in Appendix B. An eventual (currently non-existing) "base" attribute of the link is not affected by the value of "base" parameter in the registration.

5.3. Operations on the Registration Resource

This section describes how the registering endpoint can maintain the registrations that it created. The registering endpoint can be the registrant-ep or the CT. The registrations are resources of the RD.

An endpoint should not use this interface for registrations that it did not create. This is usually enforced by security policies, which in general require equivalent credentials for creation of and operations on a registration.

After the initial registration, the registering endpoint retains the returned location of the registration resource for further operations, including refreshing the registration in order to extend the lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this endpoint. The RD SHOULD continue to provide access to the registration resource after a registration time-out occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of garbage collection. If the registration resource is removed, the corresponding endpoint will need to be re-registered.

The registration resource may also be used cancel the registration using DELETE, and to perform further operations beyond the scope of this specification.

Operations on the registration resource are sensitive to reordering; Section 5.3.4 describes how order is restored.

The operations on the registration resource are described below.

5.3.1. Registration Update

The update interface is used by the registering endpoint to refresh or update its registration with an RD. To use the interface, the registering endpoint sends a POST request to the registration resource returned by the initial registration operation.

An update MAY update registration parameters like lifetime, base URI or others. Parameters that are not being changed should not be included in an update. Adding parameters that have not changed increases the size of the message but does not have any other implications. Parameters are included as query parameters in an update operation as in Section 5.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the base URI of the registration is changed in an update, relative references submitted in the original registration or later updates are resolved anew against the new base.

The registration update operation only describes the use of POST with an empty payload. Future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration (see Section 5.3.3).

The update registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+location}{?lt,base,extra-attrs*}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 90000) SHOULD be used.

base := Base URI (optional). This parameter updates the Base URI established in the original registration to a new value, and is subject to the same restrictions as in the registration.

If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the relative links present in the payload of the original registration.

If the parameter is not set in the request but was set before, the previous Base URI value is kept unmodified.

If the parameter is not set in the request and was not set before either, the source address and source port of the update request are stored as the Base URI.

extra-attrs := Additional registration

attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Note that this default behavior does not allow removing an endpoint attribute in an update. For attributes whose functionality depends on the endpoints' ability to remove them in an update, it can make sense to define a value whose presence is equivalent to the absence of a value. As an alternative, an extension can define different updating rules for their attributes. That necessitates either discovery of whether the RD is aware of that extension, or tolerating the default behavior.

Content-Format: none (no payload)

The following responses are expected on this interface:

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

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

If the registration update fails in any way, including "Not Found" and request timeouts, or if the time indicated in a Service Unavailable Max-Age/Retry-After exceeds the remaining lifetime, the registering endpoint SHOULD attempt registration again.

The following example shows how the registering endpoint resets the timeout on its registration resource at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

Figure 13: Example update of a registration

The following example shows the registering endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the registering endpoint set the following values:

* endpoint name (ep)=endpoint1

* lifetime (lt)=500

* Base URI (base)=coap://local-proxy-old.example.com

* payload of Figure 8

The initial state of the RD is reflected in the following request:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content
Payload:
<coap://local-proxy-old.example.com/sensors/temp>;
    rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
    anchor="coap://local-proxy-old.example.com/sensors/temp";
    rel=describedby
```

Figure 14: Example lookup before a change to the base address

The following example shows the registering endpoint changing the Base URI to "coaps://new.example.com:5684":

```
Req: POST /rd/4521?base=coaps://new.example.com

Res: 2.04 Changed
```

Figure 15: Example registration update that changes the base address

The consecutive query returns:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content
Payload:
<coaps://new.example.com/sensors/temp>;
    rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
    anchor="coaps://new.example.com/sensors/temp";
    rel=describedby
```

Figure 16: Example lookup after a change to the base address

5.3.2. Registration Removal

Although RD registrations have soft state and will eventually timeout after their lifetime, the registering endpoint SHOULD explicitly remove an entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP or CT -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

The following responses are expected on this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may already have been removed).

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

Figure 17: Example of a registration removal

5.3.3. Further operations

Additional operations on the registration can be specified in future documents, for example:

* Send iPATCH (or PATCH) updates ([RFC8132]) to add, remove or change the links of a registration.

* Use GET to read the currently stored set of links in a registration resource.

Those operations are out of scope of this document, and will require media types suitable for modifying sets of links.

5.3.4. Request freshness

Some security mechanisms usable with an RD allow out of order request processing, or do not even mandate replay protection at all. The RD needs to ensure that operations on the registration resource are executed in an order that does not distort the client's intentions.

This ordering of operations is expressed in terms of freshness as defined in [I-D.ietf-core-echo-request-tag]. Requests that alter a resource's state need to be fresh relative to the latest request that altered that state in a conflicting way.

An RD SHOULD determine a request's freshness, and MUST use the Echo option if it requires request freshness and can not determine the it in any other way. An endpoint MUST support the use of the Echo option. (One reason why an RD would not require freshness is when no relevant registration properties are covered by is security policies.)

5.3.4.1. Efficient use of Echo by an RD

To keep latency and traffic added by the freshness requirements to a minimum, RDs should avoid naive (sufficient but inefficient) freshness criteria.

Some simple mechanisms the RD can employ are:

* State counter. The RD can keep a monotonous counter that increments whenever a registration changes. For every registration resource, it stores the post-increment value of that resource's last change. Requests altering them need to have at least that value encoded in their Echo option, and are otherwise rejected with a 4.01 Unauthorized and the current counter value as the Echo value. If other applications on the same server use Echo as well, that encoding may include a prefix indicating that it pertains to the RD's counter.

  The value associated with a resource needs to be kept across the removal of registrations if the same registration resource is to be reused.

  The counter can be reset (and the values of removed resources forgotten) when all previous security associations are reset.

This is the "Persistent Counter" method of [I-D.ietf-core-echo-request-tag] Appendix A.

* Preemptive Echo values. The current state counter can be sent in an Echo option not only when requests are rejected with 4.01 Unauthorized, but also with successful responses. Thus, clients can be provided with Echo values sufficient for their next request on a regular basis.

  While endpoints may discard received Echo values at leisure between requests, they are encouraged to retain these values for the next request to avoid additional round trips.

* If the RD can ensure that only one security association has modifying access to any registration at any given time, and that security association provides order on the requests, that order is sufficient to show request freshness.

5.3.4.2. Examples of Echo usage

Figure 18 shows the interactions of an endpoint that has forgotten the server's latest Echo value and temporarily reduces its registration lifetime:

```
Req: POST /rd/4521?lt=7200

Res: 4.01 Unauthorized
Echo: 0x0123

(EP tries again immediately)

Req: POST /rd/4521?lt=7200
Echo: 0x0123

Res: 2.04 Changed
Echo: 0x0124

(Later the EP regains its confidence in its long-term reachability)

Req: POST /rd/4521?lt=90000
Echo: 0x0124

Res: 2.04 Changed
Echo: 0x0247
```

Figure 18: Example update of a registration

The other examples do not show Echo options for simplicity, and because they lack the context for any example values to have meaning.

6. RD Lookup

To discover the resources registered with the RD, a lookup interface must be provided. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. JSON or CBOR link format [I-D.ietf-core-links-json]) or using more advanced interfaces (e.g. supporting context or semantic based lookup) on different resources that are discovered independently.

RD Lookup allows lookups for endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, an endpoint lookup MUST return a list of endpoints and a resource lookup MUST return a list of links to resources.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

Lookup Type	Resource Type	Mandatory
Resource	core.rd-lookup-res	Mandatory
Endpoint	core.rd-lookup-ep	Mandatory

Table 1: Lookup Types

6.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD by the registrant. The links and link parameters returned by the lookup are equal to the originally submitted ones, except that the target reference is fully resolved, and that the anchor reference is fully resolved if it is present in the lookup result at all.

Links that did not have an anchor attribute in the registration are returned without an anchor attribute. Links of which href or anchor was submitted as a (full) URI are returned with the respective attribute unmodified.

The above rules allow the client to interpret the response as links without any further knowledge of the storage conventions of the RD. The RD MAY replace the registration base URIs with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

If the base URI of a registration contains a link-local address, the RD MUST NOT show its links unless the lookup was made from the link on which the registered endpoint can be reached. The RD MUST NOT include zone identifiers in the resolved URIs.

6.2. Lookup filtering

Using the Accept Option, the requester can control whether the returned list is returned in CoRE Link Format ("application/link-format", default) or in alternate content-formats (e.g. from [I-D.ietf-core-links-json]).

Multiple search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned. The RD MUST support matching with multiple search criteria.

A link matches a search criterion if it has an attribute of the same name and the same value, allowing for a trailing "*" wildcard operator as in Section 4.1 of [RFC6690]. Attributes that are defined as "relation-types" (in the link-format ABNF) match if the search value matches any of their values (see Section 4.1 of [RFC6690]; e.g. "?if=tag:example.net,2020:sensor" matches ";if="example.regname tag:example.net,2020:sensor";"). A resource link also matches a search criterion if its endpoint would match the criterion, and vice versa, an endpoint link matches a search criterion if any of its resource links matches it.

Note that "href" is a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it. Queries for resource link targets MUST be in URI form (i.e. not relative references) and are matched against a resolved link target. Queries for endpoints SHOULD be expressed in path-absolute form if possible and MUST be expressed in URI form otherwise; the RD SHOULD recognize either. The "anchor" attribute is usable for resource lookups, and, if queried, MUST be in URI form as well.

Additional query parameters "page" and "count" are used to obtain lookup results in specified increments using pagination, where count specifies how many links to return and page specifies which subset of links organized in sequential pages, each containing ’count’ links,

starting with link zero and page zero. Thus, specifying count of 10 and page of 0 will return the first 10 links in the result set (links 0-9). Count = 10 and page = 1 will return the next 'page' containing links 10-19, and so on. Unlike block-wise transfer of a compelte result set, these parameters ensure that each chunk of results can be interpreted on its own. This simplifies the processing, but can result in duplicate or missed items when coinciding with changes from the registration interface.

Endpoints that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

When resource observation is used, every time the set of matching links changes, or the content of a matching link changes, the RD sends a notification with the matching link set. The notification contains the successful current response to the given request, especially with respect to representing zero matching links (see "Success" item below).

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables: type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 4.3.

search := Search criteria for limiting the number of results (optional).

The search criteria are an associative array, expressed in a form-style query as per the URI template (see [RFC6570] Sections 2.4.2 and 3.2.8)

page := Page (optional). Parameter cannot be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of

results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

Accept: absent, application/link-format or any other indicated media type representing web links

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format" or other web link payload containing matching entries for the lookup.

The payload can contain zero links (which is an empty payload in [RFC6690] link format, but could also be "[]" in JSON based formats), indicating that no entities matched the request.

6.3. Resource lookup examples

The examples in this section assume the existence of CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example resource look-up locations discovered in Figure 5:

```
Req: GET /rd-lookup/res?rt=tag:example.org,2020:temperature

Res: 2.05 Content
Payload:
<coap://[2001:db8:3::123]:61616/temp>;
    rt="tag:example.org,2020:temperature"
```

Figure 19: Example a resource lookup

A client that wants to be notified of new resources as they show up can use observation:

```
Req: GET /rd-lookup/res?rt=tag:example.org,2020:light
Observe: 0

Res: 2.05 Content
Observe: 23
Payload: empty

(at a later point in time)

Res: 2.05 Content
Observe: 24
Payload:
<coap://[2001:db8:3::124]/west>;rt="tag:example.org,2020:light",
<coap://[2001:db8:3::124]/south>;rt="tag:example.org,2020:light",
<coap://[2001:db8:3::124]/east>;rt="tag:example.org,2020:light"
```

Figure 20: Example an observing resource lookup

The following example shows a client performing a paginated resource lookup

```
Req: GET /rd-lookup/res?page=0&count=5

Res: 2.05 Content
Payload:
<coap://[2001:db8:3::123]:61616/res/0>;ct=60,
<coap://[2001:db8:3::123]:61616/res/1>;ct=60,
<coap://[2001:db8:3::123]:61616/res/2>;ct=60,
<coap://[2001:db8:3::123]:61616/res/3>;ct=60,
<coap://[2001:db8:3::123]:61616/res/4>;ct=60

Req: GET /rd-lookup/res?page=1&count=5

Res: 2.05 Content
Payload:
<coap://[2001:db8:3::123]:61616/res/5>;ct=60,
<coap://[2001:db8:3::123]:61616/res/6>;ct=60,
<coap://[2001:db8:3::123]:61616/res/7>;ct=60,
<coap://[2001:db8:3::123]:61616/res/8>;ct=60,
<coap://[2001:db8:3::123]:61616/res/9>;ct=60
```

Figure 21: Examples of paginated resource lookup

The following example shows a client performing a lookup of all resources of all endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th response of section 5 of [RFC6690].

It demonstrates how absolute link targets stay unmodified, while relative ones are resolved:

```
Req: GET /rd-lookup/res?et=tag:example.com,2020:platform

Res: 2.05 Content
Payload:
<coap://sensor1.example.com/sensors>;ct=40;title="Sensor Index",
<coap://sensor1.example.com/sensors/temp>;rt=temperature-c;if=sensor,
<coap://sensor1.example.com/sensors/light>;rt=light-lux;if=sensor,
<http://www.example.com/sensors/t123>;rel=describedby;
    anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor1.example.com/t>;rel=alternate;
    anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor2.example.com/sensors>;ct=40;title="Sensor Index",
<coap://sensor2.example.com/sensors/temp>;rt=temperature-c;if=sensor,
<coap://sensor2.example.com/sensors/light>;rt=light-lux;if=sensor,
<http://www.example.com/sensors/t123>;rel=describedby;
    anchor="coap://sensor2.example.com/sensors/temp",
<coap://sensor2.example.com/t>;rel=alternate;
    anchor="coap://sensor2.example.com/sensors/temp"
```

Figure 22: Example of resource lookup from multiple endpoints

6.4. Endpoint lookup

The endpoint lookup returns links to and information about registration resources, which themselves can only be manipulated by the registering endpoint.

Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base; reports the registrant-ep’s address if no explicit base was given) as well as a constant resource type (rt="core.rd-ep"); the lifetime (lt) is not reported. Additional endpoint attributes are added as target attributes to their endpoint link unless their specification says otherwise.

Links to endpoints SHOULD be presented in path-absolute form or, if required, as (full) URIs. (This ensures that the output conforms to Limited Link Format as described in Appendix C.)

Base addresses that contain link-local addresses MUST NOT include zone identifiers, and such registrations MUST NOT be shown unless the lookup was made from the same link from which the registration was made.

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

An RD can report registrations in lookup whose URI scheme and authority differ from the lookup resource's. Lookup clients MUST be prepared to see arbitrary URIs as registration resources in the results and treat them as opaque identifiers; the precise semantics of such links are left to future specifications.

The following example shows a client performing an endpoint lookup limited to endpoints of endpoint type "tag:example.com,2020:platform":

```
Req: GET /rd-lookup/ep?et=tag:example.com,2020:platform

Res: 2.05 Content
Payload:
</rd/1234>;base="coap://[2001:db8:3::127]:61616";ep=node5;
    et="tag:example.com,2020:platform";ct=40;rt=core.rd-ep,
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep=node7;
    et="tag:example.com,2020:platform";ct=40;d=floor-3;
    rt=core.rd-ep
```

Figure 23: Examples of endpoint lookup

7. Security policies

The security policies that are applicable to an RD strongly depend on the application, and are not set out normatively here.

This section provides a list of aspects that applications should consider when describing their use of the RD, without claiming to cover all cases. It is using terminology of [I-D.ietf-ace-oauth-authz], in which the RD acts as the Resource Server (RS), and both registrant-eps and lookup clients act as Clients (C) with support from an Authorization Server (AS), without the intention of ruling out other (e.g. certificate / public-key infrastructure (PKI) based) schemes.

Any, all or none of the below can apply to an application. Which are relevant depends on its protection objectives.

Security policies are set by configuration of the RD, or by choice of the implementation. Lookup clients (and, where relevant, endpoints) can only trust an RD to uphold them if it is authenticated, and authorized to serve as an RD according to the application's requirements.

7.1. Endpoint name

Whenever an RD needs to provide trustworthy results to clients doing endpoint lookup, or resource lookup with filtering on the endpoint name, the RD must ensure that the registrant is authorized to use the given endpoint name. This applies both to registration and later to operations on the registration resource. It is immaterial whether the client is the registrant-ep itself or a CT is doing the registration: The RD cannot tell the difference, and CTs may use authorization credentials authorizing only operations on that particular endpoint name, or a wider range of endpoint names.

It is up to the concrete security policy to describe how endpoint name and sector are transported when certificates are used. For example, it may describe how SubjectAltName dNSName entries are mapped to endpoint and domain names.

7.1.1. Random endpoint names

Conversely, in applications where the RD does not check the endpoint name, the authorized registering endpoint can generate a random number (or string) that identifies the endpoint. The RD should then remember unique properties of the registrant, associate them with the registration for as long as its registration resource is active (which may be longer than the registration's lifetime), and require the same properties for operations on the registration resource.

Registrants that are prepared to pick a different identifier when their initial attempt (or attempts, in the unlikely case of two subsequent collisions) at registration is unauthorized should pick an identifier at least twice as long as the expected number of registrants; registrants without such a recovery options should pick significantly longer endpoint names (e.g. using UUID URNs [RFC4122]).

7.2. Entered resources

When lookup clients expect that certain types of links can only originate from certain endpoints, then the RD needs to apply filtering to the links an endpoint may register.

For example, if clients use an RD to find a server that provides firmware updates, then any registrant that wants to register (or update) links to firmware sources will need to provide suitable credentials to do so, independently of its endpoint name.

Note that the impact of having undesirable links in the RD depends on the application: if the client requires the firmware server to present credentials as a firmware server, a fraudulent link's impact is limited to the client revealing its intention to obtain updates and slowing down the client until it finds a legitimate firmware server; if the client accepts any credentials from the server as long as they fit the provided URI, the impact is larger.

An RD may also require that links are only registered if the registrant is authorized to publish information about the anchor (or even target) of the link. One way to do this is to demand that the registrant present the same credentials as a client that they'd need to present if contacted as a server at the resources' URI, which may include using the address and port that are part of the URI. Such a restriction places severe practical limitations on the links that can be registered.

As above, the impact of undesirable links depends on the extent to which the lookup client relies on the RD. To avoid the limitations, RD applications should consider prescribing that lookup clients only use the discovered information as hints, and describe which pieces of information need to be verified because they impact the application's security. A straightforward way to verify such information is to request it again from an authorized server, typically the one that hosts the target resource. That similar to what happens in Section 4.3 when the URI discovery step is repeated.

7.3. Link confidentiality

When registrants publish information in the RD that is not available to any client that would query the registrant's /.well-known/core interface, or when lookups to that interface are subject so stricter firewalling than lookups to the RD, the RD may need to limit which lookup clients may access the information.

In this case, the endpoint (and not the lookup clients) needs to be careful to check the RD's authorization. The RD needs to check any lookup client's authorization before revealing information directly (in resource lookup) or indirectly (when using it to satisfy a resource lookup search criterion).

7.4. Segmentation

Within a single RD, different security policies can apply.

One example of this are multi-tenant deployments separated by the sector (d) parameter. Some sectors might apply limitations on the endpoint names available, while others use a random identifier approach to endpoint names and place limits on the entered links based on their attributes instead.

Care must be taken in such setups to determine the applicable access control measures to each operation. One easy way to do that is to mandate the use of the sector parameter on all operations, as no credentials are suitable for operations across sector borders anyway.

7.5. First-Come-First-Remembered: A default policy

The First-Come-First-Remembered policy is provided both as a reference example for a security policy definition, and as a policy that implementations may choose to use as default policy in absence of other configuration. It is designed to enable efficient discovery operations even in ad-hoc settings.

Under this policy, the RD accepts registrations for any endpoint name that is not assigned to an active registration resource, and only accepts registration updates from the same endpoint. The policy is minimal in that towards lookup clients it does not make any of the claims of Section 7.2 and Section 7.3, and its claims on Section 7.1 are limited to the lifetime of that endpoint's registration. It does, however, guarantee towards any endpoint that for the duration of its registration, its links will be discoverable on the RD.

When a registration or operation is attempted, the RD MUST determine the client's subject name or public key:

* If the client's credentials indicate any subject name that is certified by any authority which the RD recognizes (which may be the system's trust anchor store), all such subject names are stored. With CWT or JWT based credentials (as common with ACE), the Subject (sub) claim is stored as a single name, if it exists. With X.509 certificates, the Common Name (CN) and the complete list of SubjectAltName entries are stored. In both cases, the authority that certified the claim is stored along with the subject, as the latter may only be locally unique.

* Otherwise, if the client proves possession of a private key, the matching public key is stored. This applies both to raw public keys and to the public keys indicated in certificates that failed the above authority check.

* If neither is present, a reference to the security session itself is stored. With (D)TLS, that is the connection itself, or the session resumption information if available. With OSCORE, that is the security context.

As part of the registration operation, that information is stored along with the registration resource.

The RD MUST accept all registrations whose registration resource is not already active, as long as they are made using a security layer supported by the RD.

Any operation on a registration resource, including registrations that lead to an existing registration resource, MUST be rejected by the RD unless all the stored information is found in the new request's credentials.

Note that even though subject names are compared in this policy, they are never directly compared to endpoint names, and an endpoint can not expect to "own" any particular endpoint name outside of an active registration -- even if a certificate says so. It is an accepted shortcoming of this approach that the endpoint has no indication of whether the RD remembers it by its subject name or public key; recognition by subject happens on a best-effort base (given the RD may not recognize any authority). Clients MUST be prepared to pick a different endpoint name when rejected by the RD initially or after a change in their credentials; picking an endpoint name as per Section 7.1.1 is an easy option for that.

For this policy to be usable without configuration, clients should not set a sector name in their registrations. An RD can set a default sector name for registrations accepted under this policy, which is useful especially in a segmented setup where different policies apply to different sectors. The configuration of such a behavior, as well as any other configuration applicable to such an RD (i.e. the set of recognized authorities) is out of scope for this document.

8. Security Considerations

The security considerations as described in Section 5 of [RFC8288] and Section 6 of [RFC6690] apply. The "/.well-known/core" resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252].

Access that is limited or affects sensitive data SHOULD be protected, e.g. using (D)TLS or OSCORE ([RFC8613]; which aspects of the RD this affects depends on the security policies of the application (see Section 7).

8.1. Discovery

Most steps in discovery of the RD, and possibly its resources, are not covered by CoAP's security mechanisms. This will not endanger the security properties of the registrations and lookup itself (where the client requires authorization of the RD if it expects any security properties of the operation), but may leak the client's intention to third parties, and allow them to slow down the process.

To mitigate that, clients can retain the RD's address, use secure discovery options like configured addresses, and send queries for RDs in a very general form ("?rt=core.rd*" rather than "?rt=core.rd-lookup-ep").

8.2. Endpoint Identification and Authentication

An Endpoint (name, sector) pair is unique within the set of endpoints registered by the RD. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint on an RD SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security.

Consider the following threat: two devices A and B are registered at a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the DTLS exchange. Then, it specifies the endpoint name of device B as the name of its own endpoint in device A. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Endpoint authorization needs to be checked on registration and registration resource operations independently of whether there are configured requirements on the credentials for a given endpoint name (and sector; Section 7.1) or whether arbitrary names are accepted (Section 7.1.1).

Simple registration could be used to circumvent address-based access control: An attacker would send a simple registration request with the victim's address as source address, and later look up the victim's /.well-known/core content in the RD. Mitigation for this is recommended in Section 5.1.

The registration resource path is visible to any client that is allowed endpoint lookup, and can be extracted by resource lookup clients as well. The same goes for registration attributes that are shown as target attributes or lookup attributes. The RD needs to consider this in the choice of registration resource paths, and administrators or endpoint in their choice of attributes.

8.3. Access Control

Access control SHOULD be performed separately for the RD registration and Lookup API paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the sector, endpoint or resource level.

The precise access controls necessary (and the consequences of failure to enforce them) depend on the protection objectives of the application and the security policies (Section 7) derived from them.

8.4. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly amplify and distribute a DoS attack as UDP does not require return routability check. Since RD lookup responses can be significantly larger than requests, RDs are prone to this.

[RFC7252] describes this at length in its Section 11.3, including some mitigation by using small block sizes in responses. The upcoming [I-D.ietf-core-echo-request-tag] updates that by describing a source address verification mechanism using the Echo option.

[If this document is published together with or after I-D.ietf-core-echo-request-tag, the above paragraph is replaced with the following:

[RFC7252] describes this at length in its Section 11.3, and [I-D.ietf-core-echo-request-tag] (which updates the former) recommends using the Echo option to verify the request's source address.

]

8.5. Skipping freshness checks

When RD based applications are built in which request freshness checks are not performed, these concerns need to be balanced:

* When alterations to registration attributes are reordered, an attacker may create any combination of attributes ever set, with the attack difficulty determined by the security layer's replay properties.

  For example, if Figure 18 were conducted without freshness assurances, an attacker could later reset the lifetime back to 7200. Thus, the device is made unreachable to lookup clients.

* When registration updates without query parameters (which just serve to restart the lifetime) can be reordered, an attacker can use intercepted messages to give the appearance of the device being alive to the RD.

  This is unacceptable when when the RD's security policy promises reachability of endpoints (e.g. when disappearing devices would trigger further investigation), but may be acceptable with other policies.

9. IANA Considerations

9.1. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values sub-registry of the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690]:

Value	Description	Reference
core.rd	Directory resource of an RD	RFCTHIS Section 4.3
core.rd-lookup-res	Resource lookup of an RD	RFCTHIS Section 4.3
core.rd-lookup-ep	Endpoint lookup of an RD	RFCTHIS Section 4.3
core.rd-ep	Endpoint resource of an RD	RFCTHIS Section 6

Table 2

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the sub-registry "IPv6 Neighbor Discovery Option Formats" of the "Internet Control Message Protocol version 6 (ICMPv6) Parameters" registry:

* Resource Directory Address Option (TBD38)

[The RFC editor is asked to replace TBD38 with the assigned number in the document; the value 38 is suggested.]

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include

* the human readable name of the parameter,

* the short name as used in query parameters or target attributes,

* indication of whether it can be passed as a query parameter at registration of endpoints, as a query parameter in lookups, or be expressed as a target attribute,

* syntax and validity requirements if any,

* a description,

* and a link to reference documentation.

The query parameter MUST be both a valid URI query key [RFC3986] and a token as used in [RFC8288].

The description must give details on whether the parameter can be updated, and how it is to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

Full name	Short	Validity	Use	Description
Endpoint Name	ep	Unicode*	RLA	Name of the endpoint
Lifetime	lt	1-4294967295	R	Lifetime of the registration in seconds
Sector	d	Unicode*	RLA	Sector to which this endpoint belongs
Registration Base URI	base	URI	RLA	The scheme, address and port and path at which this server is available
Page	page	Integer	L	Used for pagination
Count	count	Integer	L	Used for pagination
Endpoint Type	et	Section 9.3.1	RLA	Semantic type of the endpoint (see Section 9.4)

Table 3: RD Parameters

(Short: Short name used in query parameters or target attributes. Validity: Unicode* = 63 Bytes of UTF-8 encoded Unicode, with no control characters as per Section 5. Use: R = used at registration, L = used at lookup, A = expressed in target attribute.)

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document. All their reference documentation entries point to this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (E.g. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used target attributes (For

example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] target attribute).

9.3.1. Full description of the "Endpoint Type" RD Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type sub-registry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, RDs implementing this specification automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called ’"Endpoint Type" (et=) RD Parameter values’. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

* The values MUST be related to the purpose described in Section 9.3.1.

* The registered values MUST conform to the ABNF reg-rel-type definition of [RFC6690] and MUST NOT be a URI.

* It is recommended to use the period "." character for segmentation.

The registry initially contains one value:

* "core.rd-group": An application group as described in Appendix A.

9.5. Multicast Address Registration

IANA is asked to assign the following multicast addresses for use by CoAP nodes:

IPv4 -- "all CoRE Resource Directories" address MCD2 (suggestion: 224.0.1.189), from the "IPv4 Multicast Address Space Registry". As the address is used for discovery that may span beyond a single network, it has come from the Internetwork Control Block (224.0.1.x) [RFC5771].

IPv6 -- "all CoRE Resource Directories" address MCD1 (suggestions FF0X::FE), from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.

[The RFC editor is asked to replace MCD1 and MCD2 with the assigned addresses throughout the document.]

9.6. Well-Known URIs

IANA is asked to permanently register the URI suffix "rd" in the "Well-Known URIs" registry. The change controller is the IETF, this document is the reference.

9.7. Service Names and Transport Protocol Port Number Registry

IANA is asked to enter four new items into the Service Names and Transport Protocol Port Number Registry:

* Service name: "core-rd", Protocol: "udp", Description: "Resource Directory accessed using CoAP"

* Service name "core-rd-dtls", Protocol: "udp", Description: "Resource Directory accessed using CoAP over DTLS"

* Service name: "core-rd", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TCP"

* Service name "core-rd-tls", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TLS"

All in common have this document as their reference.

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LwM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the RD with a CoAP interface to facilitate the installation and start-up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address as described in Appendix A. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative.

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infrastructure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and the sensor. The addresses shown in Table 4 below stand in for these in the following examples.

Name	IPv6 address
luminary1	2001:db8:4::1
luminary2	2001:db8:4::2
Presence sensor	2001:db8:4::3
RD	2001:db8:4::ff

Table 4: Addresses used in the examples

In Section 10.1.2 the use of RD during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have rt=tag:example.com,2020:light, and the presence sensor has rt=tag:example.com,2020:p-sensor. The endpoints have names which are relevant to the light installation manager. In this case luminary1, luminary2, and the presence sensor are located in room 2-4-015, where luminary1 is located at the window and luminary2 and the presence sensor are located at the door. The endpoint names reflect this physical location. The middle, left and right lamps are accessed via path /light/middle, /light/left, and /light/right respectively. The identifiers relevant to the RD are shown in Table 5 below:

Name	endpoint	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/middle	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/right	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/left	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/middle	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/right	tag:example.com,2020:light
Presence sensor	ps_R2-4-015_door	/ps	tag:example.com,2020:p-sensor

Table 5: RD identifiers

It is assumed that the CT has performed RD discovery and has received a response like the one in the Section 4.3 example.

The CT inserts the endpoints of the luminaries and the sensor in the RD using the registration base URI parameter (base) to specify the interface address:

```
Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=lm_R2-4-015_wndw&base=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/4521

Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=lm_R2-4-015_door&base=coap://[2001:db8:4::2]&d=R2-4-015
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/4522

Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=ps_R2-4-015_door&base=coap://[2001:db8:4::3]&d=R2-4-015
Payload:
</ps>;rt="tag:example.com,2020:p-sensor"

Res: 2.01 Created
Location-Path: /rd/4523
```

Figure 24: Example of registrations a CT enters into an RD

The sector name d=R2-4-015 has been added for an efficient lookup because filtering on "ep" name is more awkward. The same sector name is communicated to the two luminaries and the presence sensor by the CT.

The group is specified in the RD. The base parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, the resources supported by all group members are published.

```
Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::1]
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/501
```

Figure 25: Example of a multicast group a CT enters into an RD

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its sector and being configured to join any group containing lights, searches for candidate groups and joins them:

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
  ?d=R2-4-015&et=core.rd-group&rt=light

Res: 2.05 Content
Payload:
</rd/501>;ep=grp_R2-4-015;et=core.rd-group;
          base="coap://[ff05::1]";rt=core.rd-ep
```

Figure 26: Example of a lookup exchange to find suitable multicast addresses

From the returned base parameter value, the luminary learns the multicast address of the multicast group.

The presence sensor can learn the presence of groups that support resources with rt=tag:example.com,2020:light in its own sector by sending the same request, as used by the luminary. The presence sensor learns the multicast address to use for sending messages to the luminaries.

10.2. OMA Lightweight M2M (LwM2M)

OMA LwM2M is a profile for device services based on CoAP, providing interfaces and operations for device management and device service enablement.

An LwM2M server is an instance of an LwM2M middleware service layer, containing an RD ([LwM2M] page 36f).

That RD only implements the registration interface, and no lookup is implemented. Instead, the LwM2M server provides access to the registered resources, in a similar way to a reverse proxy.

The location of the LwM2M Server and RD URI path is provided by the LwM2M Bootstrap process, so no dynamic discovery of the RD is used. LwM2M Servers and endpoints are not required to implement the /.well-known/core resource.

11. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, Jan Newmarch, Matthias Kovatsch, Jaime Jimenez and Ted Lemon have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the RD concepts were originally developed.

12. Changelog

changes from -27 to -28

* Security policies / link confidentiality: Point out the RD's obligations that follow from such a policy.

* Simple registration: clarify term "regular registration" by introducing it along with the reference to Section 5

* Wording fix in first-come-first-remembered

* Wording fixes in RD definition

* Capitalization: Consistently using "registration resource"

changes from -26 to -27

* In general, this addresses the points that were pointed out in https://mailarchive.ietf.org/arch/msg/core/xWLomwwhovkU-CPGNxnvs40BhaM/ as having "evolved from the review comments being discussed in the interim meetings", and the review comments from Esko Dijk that were largely entangled in these points.

* Relaxation of the serialization rules for link-format

  The interpretation of RFC6690 used in Appendix B.4 was shown to be faulty. Along with a correction, the common implementations of link-format were surveyed again and it was found that the only one

that employed the faulty interpretation can still safely be upgraded. These were removed from the set considered for Limited Link Format, making the set of valid Limited Link Format documents larger.

As a consequence, the prescribed serialization of RD output can be roughly halved in bytes.

There might be additional usage patterns that are possible with the new set of constraints, but there is insufficient implementation and deployment experience with them to warrant a change changes on that front at this point. The specification can later be extended compatibly to allow these cases and drop the requirement of Limited Link Format.

* Add Request freshness subsection

  It is now recommended (with security considerations on consequences of not doing it) to require ordering of RD operations.

  The Echo mechanism (previously suggested in various places but never exclusively) is the one prescribed way of getting this ordering, making the echo-request-tag reference normative.

* Improved expression about when an RD needs to verify simple registration.

  The simple wording missed the authorization part, and did not emphasize that this is a per-deployment property.

* Point out the non-atomic properties of paginated access.

* Clarification around impl-info reference.

* Inconsistencies and extraneous quotings removed from examples.

changes from -25 to -26

* Security policies:

  - The First-Come-First-Remembered policy is added as an example and a potential default behavior.

  - Clarify that the mapping between endpoint names and subject fields is up to a policy that defines reliance on names, and give an example.

- Random EP names: Point that multiple collisions are possible but unlikely.

- Add pointers to policies:

  o RD replication: Point out that policies may limit that.

  o Registration: Reword (ep, d) mapping to a previous registration's resource that could have been read as another endpoint taking over an existing registration.

- Clarify that the security policy is a property of the RD the any client may need to verify by checking the RD's authorization.

- Clarify how information from an untrusted RD can be verified

- Remove speculation about how in detail ACE scopes are obtained.

* Security considerations:

  - Generalize to all current options for security layers usable with CoAP (OSCORE was missing as the text predated RFC8613)

  - Relax the previous SHOULD on secure access to SHOULD where protection is indicated by security policies (bringing the text in line with the -25 changes)

  - Point out that failure to follow the security considerations has implications depending on the protection objective described with the security policies

  - Shorten amplification mitigation

  - Add note about information in Registration Resource path.

  - Acknowledge that most host discovery operations are not secured; mention consequences and mitigation.

* Abstract, introduction: removed "or disperse networks"

* RD discovery:

  - Drop the previously stated assumption that RDAO and any DHCP options would only be used together with SLAAC and DHCP for address configuration, respectivly.

- Give concrete guidance for address selection based on RFC6724 when responding to multicasts

- RDAO:

  o Clarify that it is an option for RAs and not other ND messages.

  o Change Lifetime from 16-bit minutes to 32-bit seconds and swap it with Reserved (aligning it with RDNSS which it shares other properties as well).

- Point out that clients may need to check RD authorization already in last discovery step

* Registration:

  - Wording around "mostly mandatory" has been improved, conflicts clarified and sector default selection adjusted.

* Simple registration: Rather than coopting POSTs to /.well-known/core, a new resource /.well-known/rd is registered. A historical note in the text documents the change.

* Examples:

  - Use example URIs rather than unclear reg names (unless it’s RFC6690 examples, which were kept for continuity)

  - The LwM2M example was reduced from an outdated explanation of the complete LwM2M model to a summary of how RD is used in there, with a reference to the current specification.

  - Luminary example: Explain example addresses

  - Luminary example: Drop reference to coap-group mechanism that’s becoming obsolete, and thus also to RFC7390

  - Multicast addresses in the examples were changed from ff35:30:2001:db8::x to ff35:30:2001:db8:f1::8000:x; the 8000 is to follow RFC 3307, and the f1 is for consistency with all the other example addresses where 2001:db8::/32 is subnetted to 2001:db8:x::/48 by groups of internally consistent examples.

* Use case text enhancements

  - Home and building automation: Tie in with RD

- M2M: Move system design paragraph towards the topic of reusability.

* Various editorial fixes in response to Gen-ART and IESG reviews.

* Rename 'Full description of the "Endpoint Type" Registration Parameter' section to '... RD Parameter'

* Error handling: Place a SHOULD around the likely cases, and make the previous "MUST to the best of their capabilities" a "must".

* impl-info: Add note about the type being WIP

* Interaction tables: list CTs as possible initiators where applicable

* Registration update: Relax requirement to not send parameters needlessly

* Terminology: Clarify that the CTs' installation events can occur multiple times.

* Promote RFCs 7252, 7230 and 8288 to normative references

* Moved Christian Amsuess to first author

changes from -24 to -25

* Large rework of section 7 (Security policies)

  Rather than prescribing which data in the RD _is_ authenticated (and how), it now describes what applications built on an RD _can_ choose to authenticate, show possibilities on how to do it and outline what it means for clients.

  This addresses Russ' Genart review points on details in the text in a rather broad fashion. That is because the discussion on the topic inside the WG showed that that text on security has been driven more review-by-review than by an architectural plan of the authors and WG.

* Add concrete suggestions (twice as long as registrant number with retries, or UUIDs without) for random endpoint names

* Point out that simple registration can have faked origins, RECOMMEND mitigation when applicable and suggest the Echo mechanism to implement it.

* Reference existing and upcoming specifications for DDOS mitigation in CoAP.

* Explain the provenance of the example's multicast address.

* Make "SHOULD" of not manipulating foreign registrations a "should" and explain how it is enforced

* Clarify application of RFC6570 to search parameters

* Syntactic fixes in examples

* IANA:

  - Don't announce expected number of registrations (goes to write-up)

  - Include syntax as part of a field's validity in entry requirements

* Editorial changes

  - Align wording between abstract and introduction

  - Abbreviation normalization: "ER model", "RD"

  - RFC8174 boilerplate update

  - Minor clarity fixes

  - Markup and layouting

changes from -23 to -24

* Discovery using DNS-SD added again

* Minimum lifetime (lt) reduced from 60 to 1

* References added

* IANA considerations

  - added about .well-known/core resource

  - added DNS-SD service names

  - made RDAO option number a suggestion

- added "reference" field to endpoint type registry

* Lookup: mention that anchor is a legitimate lookup attribute

* Terminology and example fixes

* Layout fixes, esp. the use of non-ASCII characters in figures

changes from -22 to -23

* Explain that updates can not remove attributes

* Typo fixes

changes from -21 to -22

* Request a dedicated IPv4 address from IANA (rather than sharing with All CoAP nodes)

* Fix erroneous examples

* Editorial changes

  - Add figure numbers to examples

  - Update RD parameters table to reflect changes of earlier versions in the text

  - Typos and minor wording

changes from -20 to -21

(Processing comments during WGLC)

* Defer outdated description of using DNS-SD to find an RD to the defining document

* Describe operational conditions in automation example

* Recommend particular discovery mechanisms for some managed network scenarios

changes from -19 to -20

(Processing comments from the WG chair review)

* Define the permissible characters in endpoint and sector names

* Express requirements on NAT situations in more abstract terms

* Shifted heading levels to have the interfaces on the same level

* Group instructions for error handling into general section

* Simple Registration: process reflowed into items list

* Updated introduction to reflect state of CoRE in general, reference RFC7228 (defining "constrained") and use "IoT" term in addition to "M2M"

* Update acknowledgements

* Assorted editorial changes

  - Unify examples style

  - Terminology: RDAO defined and not only expanded

  - Add CT to Figure 1

  - Consistency in the use of the term "Content Format"

changes from -18 to -19

* link-local addresses: allow but prescribe split-horizon fashion when used, disallow zone identifiers

* Remove informative references to documents not mentioned any more

changes from -17 to -18

* Rather than re-specifying link format (Modernized Link Format), describe a Limited Link Format that’s the uncontested subset of Link Format

* Acknowledging the -17 version as part of the draft

* Move "Read endpoint links" operation to future specification like PATCH

* Demote links-json to an informative reference, and removed them from exchange examples

* Add note on unusability of link-local IP addresses, and describe mitigation.

* Reshuffling of sections: Move additional operations and endpoint lookup back from appendix, and groups into one

* Lookup interface tightened to not imply applicability for non link-format lookups (as those can have vastly different views on link cardinality)

* Simple registration: Change sequence of GET and POST-response, ensuring unsuccessful registrations are reported as such, and suggest how devices that would have required the inverse behavior can still cope with it.

* Abstract and introduction reworded to avoid the impression that resources are stored in full in the RD

* Simplify the rules governing when a registration resource can or must be changed.

* Drop a figure that has become useless due to the changes of and -13 and -17

* Wording consistency fixes: Use "Registrations" and "target attributes"

* Fix incorrect use of content negotiation in discovery interface description (Content-Format -> Accept)

* State that the base attribute value is part of endpoint lookup even when implicit in the registration

* Update references from RFC5988 to its update RFC8288

* Remove appendix on protocol-negotiation (which had a note to be removed before publication)

changes from -16 to -17

(Note that -17 is published as a direct follow-up to -16, containing a single change to be discussed at IETF103)

* Removed groups that are enumerations of registrations and have dedicated mechanism

* Add groups that are enumerations of shared resources and are a special case of endpoint registrations

changes from -15 to -16

* Recommend a common set of resources for members of a group

* Clarified use of multicast group in lighting example

* Add note on concurrent registrations from one EP being possible but not expected

* Refresh web examples appendix to reflect current use of Modernized Link Format

* Add examples of URIs where Modernized Link Format matters

* Editorial changes

changes from -14 to -15

* Rewrite of section "Security policies"

* Clarify that the "base" parameter text applies both to relative references both in anchor and href

* Renamed "Registree-EP" to Registrant-EP"

* Talk of "relative references" and "URIs" rather than "relative" and "absolute" URIs. (The concept of "absolute URIs" of [RFC3986] is not needed in RD).

* Fixed examples

* Editorial changes

changes from -13 to -14

* Rename "registration context" to "registration base URI" (and "con" to "base") and "domain" to "sector" (where the abbreviation "d" stays for compatibility reasons)

* Introduced resource types core.rd-ep and core.rd-gp

* Registration management moved to appendix A, including endpoint and group lookup

* Minor editorial changes

  - PATCH/iPATCH is clearly deferred to another document

  - Recommend against query / fragment identifier in con=

- Interface description lists are described as illustrative
- Rewording of Simple Registration

* Simple registration carries no error information and succeeds immediately (previously, sequence was unspecified)
* Lookup: href are matched against resolved values (previously, this was unspecified)
* Lookup: lt are not exposed any more
* con/base: Paths are allowed
* Registration resource locations can not have query or fragment parts
* Default life time extended to 25 hours
* clarified registration update rules
* lt-value semantics for lookup clarified.
* added template for simple registration

changes from -12 to -13

* Added "all resource directory" nodes MC address
* Clarified observation behavior
* version identification
* example rt= and et= values
* domain from figure 2
* more explanatory text
* endpoints of a groups hosted by different RD
* resolve RFC6690-vs-8288 resolution ambiguities:
  - require registered links not to be relative when using anchor
  - return absolute URIs in resource lookup

changes from -11 to -12

* added Content Model section, including ER diagram

* removed domain lookup interface; domains are now plain attributes of groups and endpoints

* updated chapter "Finding a Resource Directory"; now distinguishes configuration-provided, network-provided and heuristic sources

* improved text on: atomicity, idempotency, lookup with multiple parameters, endpoint removal, simple registration

* updated LWM2M description

* clarified where relative references are resolved, and how context and anchor interact

* new appendix on the interaction with RFCs 6690, 5988 and 3986

* lookup interface: group and endpoint lookup return group and registration resources as link targets

* lookup interface: search parameters work the same across all entities

* removed all methods that modify links in an existing registration (POST with payload, PATCH and iPATCH)

* removed plurality definition (was only needed for link modification)

* enhanced IANA registry text

* state that lookup resources can be observable

* More examples and improved text

changes from -09 to -10

* removed "ins" and "exp" link-format extensions.

* removed all text concerning DNS-SD.

* removed inconsistency in RDAO text.

* suggestions taken over from various sources

* replaced "Function Set" with "REST API", "base URI", "base path"

* moved simple registration to registration section

changes from -08 to -09

* clarified the "example use" of the base RD resource values /rd, /rd-lookup, and /rd-group.

* changed "ins" ABNF notation.

* various editorial improvements, including in examples

* clarifications for RDAO

changes from -07 to -08

* removed link target value returned from domain and group lookup types

* Maximum length of domain parameter 63 bytes for consistency with group

* removed option for simple POST of link data, don’t require a .well-known/core resource to accept POST data and handle it in a special way; we already have /rd for that

* add IPv6 ND Option for discovery of an RD

* clarify group configuration section 6.1 that endpoints must be registered before including them in a group

* removed all superfluous client-server diagrams

* simplified lighting example

* introduced Commissioning Tool

* RD-Look-up text is extended.

changes from -06 to -07

* added text in the discovery section to allow content format hints to be exposed in the discovery link attributes

* editorial updates to section 9

* update author information

* minor text corrections

Changes from -05 to -06

* added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

* added Update Endpoint Links using PATCH

* http access made explicit in interface specification

* Added http examples

Changes from -03 to -04:

* Added http response codes

* Clarified endpoint name usage

* Add application/link-format+cbor content-format

Changes from -02 to -03:

* Added an example for lighting and DNS integration

* Added an example for RD use in OMA LWM2M

* Added Read Links operation for link inspection by endpoints

* Expanded DNS-SD section

* Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

* Added a catalogue use case.

* Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.

* Additional examples section added for more complex use cases.

* New DNS-SD mapping section.

* Added text on endpoint identification and authentication.

* Error code 4.04 added to Registration Update and Delete requests.

* Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

* Removed the ETag validation feature.

* Place holder for the DNS-SD mapping section.

* Explicitly disabled GET or POST on returned Location.

* New registry for RD parameters.

* Added support for the JSON Link Format.

* Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

* Updated the version and date.

Changes from -04 to -05:

* Restricted Update to parameter updates.

* Added pagination support for the Lookup interface.

* Minor editing, bug fixes and reference updates.

* Added group support.

* Changed rt to et for the registration and update interface.

Changes from -03 to -04:

* Added the ins= parameter back for the DNS-SD mapping.

* Integrated the Simple Directory Discovery from Carsten.

* Editorial improvements.

* Fixed the use of ETags.

* Fixed tickets 383 and 372

Changes from -02 to -03:

* Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.

* Updated REST interface descriptions to use RFC6570 URI Template format.

* Introduced an improved RD Lookup design as its own function set.

* Improved the security considerations section.

* Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

* Added a terminology section.

* Changed the inclusion of an ETag in registration or update to a MAY.

* Added the concept of an RD Domain and a registration parameter for it.

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

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

13. References

13.1. Normative References

[I-D.ietf-core-echo-request-tag]
Amsüss, C., Mattsson, J. P., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", Work in Progress, Internet-Draft, draft-ietf-core-echo-request-tag-12, 1 February 2021, <https://www.ietf.org/archive/id/draft-ietf-core-echo-request-tag-12.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

13.2. Informative References

[ER] Chen, P., "The entity-relationship model--toward a unified view of data", DOI 10.1145/320434.320440, ACM Transactions on Database Systems Vol. 1, pp. 9-36, March 1976, <https://doi.org/10.1145/320434.320440>.

[I-D.bormann-t2trg-rel-impl]
Bormann, C., "impl-info: A link relation type for disclosing implementation information", Work in Progress, Internet-Draft, draft-bormann-t2trg-rel-impl-02, 27 September 2020, <https://www.ietf.org/archive/id/draft-bormann-t2trg-rel-impl-02.txt>.

[I-D.hartke-t2trg-coral]
Hartke, K., "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-hartke-t2trg-coral-09, 8 July 2019, <https://www.ietf.org/archive/id/draft-hartke-t2trg-coral-09.txt>.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-37, 4 February 2021, <https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-37.txt>.

[I-D.ietf-core-links-json]
LI, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", Work in Progress, Internet-Draft, draft-ietf-core-links-json-10, 26 February 2018, <https://www.ietf.org/archive/id/draft-ietf-core-links-json-10.txt>.

[I-D.ietf-core-rd-dns-sd]
Stok, P. V. D., Koster, M., and C. Amsüss, "CoRE Resource Directory: DNS-SD mapping", Work in Progress, Internet-Draft, draft-ietf-core-rd-dns-sd-05, 7 July 2019, <https://www.ietf.org/archive/id/draft-ietf-core-rd-dns-sd-05.txt>.

[I-D.silverajan-core-coap-protocol-negotiation]
Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", Work in Progress, Internet-Draft, draft-silverajan-core-coap-protocol-negotiation-09, 2 July 2018, <https://www.ietf.org/archive/id/draft-silverajan-core-coap-protocol-negotiation-09.txt>.

[LwM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification: Transport Bindings (Candidate Version 1.1)", 12 June 2018,

<https://openmobilealliance.org/RELEASE/LightweightM2M/V1_1-20180612-C/OMA-TS-LightweightM2M_Transport-V1_1-20180612-C.pdf>.

[RFC3306] Haberman, B. and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses", RFC 3306, DOI 10.17487/RFC3306, August 2002, <https://www.rfc-editor.org/info/rfc3306>.

[RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, DOI 10.17487/RFC3849, July 2004, <https://www.rfc-editor.org/info/rfc3849>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5771] Cotton, M., Vegoda, L., and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments", BCP 51, RFC 5771, DOI 10.17487/RFC5771, March 2010, <https://www.rfc-editor.org/info/rfc5771>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/info/rfc6724>.

[RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <https://www.rfc-editor.org/info/rfc6775>.

[RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <https://www.rfc-editor.org/info/rfc6874>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

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

[RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <https://www.rfc-editor.org/info/rfc8106>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

Appendix A. Groups Registration and Lookup

The RD-Groups usage pattern allows announcing application groups inside an RD.

Groups are represented by endpoint registrations. Their base address is a multicast address, and they SHOULD be entered with the endpoint type "core.rd-group". The endpoint name can also be referred to as a group name in this context.

The registration is inserted into the RD by a Commissioning Tool, which might also be known as a group manager here. It performs third party registration and registration updates.

The links it registers SHOULD be available on all members that join the group. Depending on the application, members that lack some resource MAY be permissible if requests to them fail gracefully.

The following example shows a CT registering a group with the name "lights" which provides two resources. The directory resource path /rd is an example RD location discovered in a request similar to Figure 5. The group address in the example is constructed from [RFC3849]’s reserved 2001:db8:: prefix as a unicast-prefix based site-local address (see [RFC3306].

```
Req: POST coap://rd.example.com/rd?ep=lights&et=core.rd-group
                            &base=coap://[ff35:30:2001:db8:f1::8000:1]
Content-Format: 40
Payload:
</light>;rt="tag:example.com,2020:light";
     if="tag:example.net,2020:actuator",
</color-temperature>;if="tag:example.net,2020:parameter";u=K

Res: 2.01 Created
Location-Path: /rd/12
```

Figure 27: Example registration of a group

In this example, the group manager can easily permit devices that have no writable color-temperature to join, as they would still respond to brightness changing commands. Had the group instead contained a single resource that sets brightness and color temperature atomically, endpoints would need to support both properties.

The resources of a group can be looked up like any other resource, and the group registrations (along with any additional registration parameters) can be looked up using the endpoint lookup interface.

The following example shows a client performing an endpoint lookup for all groups.

```
Req: GET /rd-lookup/ep?et=core.rd-group

Res: 2.05 Content
Payload:
</rd/12>;ep=lights&et=core.rd-group;
         base="coap://[ff35:30:2001:f1:db8::8000:1]";rt=core.rd-ep
```

Figure 28: Example lookup of groups

The following example shows a client performing a lookup of all resources of all endpoints (groups) with et=core.rd-group.

```
Req: GET /rd-lookup/res?et=core.rd-group

Res: 2.05 Content
Payload:
<coap://[ff35:30:2001:db8:f1::8000:1]/light>;
     rt="tag:example.com,2020:light";
     if="tag:example.net,2020:actuator",
<coap://[ff35:30:2001:db8:f1::8000:1]/color-temperature>;
     if="tag:example.net,2020:parameter";u=K,
```

Figure 29: Example lookup of resources inside groups

Appendix B. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC8288] which defines Link header fields, and [RFC7252] providing the transport). This appendix summarizes the mechanisms and semantics at play from an entry in "/.well-known/core" to a resource lookup.

This text is primarily aimed at people entering the field of Constrained Restful Environments from applications that previously did not use web mechanisms.

B.1. A simple example

Let’s start this example with a very simple host, "2001:db8:f0::1". A client that follows classical CoAP Discovery ([RFC7252] Section 7), sends the following multicast request to learn about neighbours supporting resources with resource-type "temperature".

The client sends a link-local multicast:

```
Req: GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature

Res: 2.05 Content
Payload:
</sensors/temp>;rt=temperature;ct=0
```

Figure 30: Example of direct resource discovery

where the response is sent by the server, "[2001:db8:f0::1]:5683".

While the client -- on the practical or implementation side -- can just go ahead and create a new request to "[2001:db8:f0::1]:5683" with Uri-Path: "sensors" and "temp", the full resolution steps for insertion into and retrieval from the RD without any shortcuts are:

B.1.1. Resolving the URIs

The client parses the single returned record. The link’s target (sometimes called "href") is ""/sensors/temp"", which is a relative URI that needs resolving. The base URI <coap://[ff02::fd]:5683/.well-known/core> is used to resolve the reference /sensors/temp against.

The Base URI of the requested resource can be composed from the options of the CoAP GET request by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into ""coap://[2001:db8:f0::1]/.well-known/core"".

Because ""/sensors/temp"" starts with a single slash, the record's target is resolved by replacing the path ""/.well-known/core"" from the Base URI (section 5.2 [RFC3986]) with the relative target URI ""/sensors/temp"" into ""coap://[2001:db8:f0::1]/sensors/temp"".

B.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content format is text/plain (ct=0).

A relation in a web link is a three-part statement that specifies a named relation between the so-called "context resource" and the target resource, like "_This page_ has _its table of contents_ at _/toc.html_". In link format documents, there is an implicit "host relation" specified with default parameter: rel="hosts".

In our example, the context resource of the link is implied to be "coap://[2001:db8:f0::1]" by the default value of the anchor (see Appendix B.4). A full English expression of the "host relation" is:

'"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/sensors/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'

B.2. A slightly more complex example

Omitting the "rt=temperature" filter, the discovery query would have given some more records in the payload:

```
Req: GET coap://[ff02::fd]:5683/.well-known/core

Res: 2.05 Content
Payload:
</sensors/temp>;rt=temperature;ct=0,
</sensors/light>;rt=light-lux;ct=0,
</t>;anchor="/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;anchor="/sensors/temp";
    rel=describedby
```

Figure 31: Extended example of direct resource discovery

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the Base URI of the request and is thus resolved to ""coap://[2001:db8:f0::1]/sensors/temp"". That is the context resource of the link, so the "rel" statement is not about the target and the Base URI any more, but about the target and the resolved URI. Thus, the third record could be read as ""coap://[2001:db8:f0::1]/sensors/temp" has an alternate representation at "coap://[2001:db8:f0::1]/t"".

Following the same resolution steps, the fourth record can be read as ""coap://[2001:db8:f0::1]/sensors/temp" is described by "http://www.example.com/sensors/t123"".

B.3. Enter the Resource Directory

The RD tries to carry the semantics obtainable by classical CoAP discovery over to the resource lookup interface as faithfully as possible.

For the following queries, we will assume that the simple host has used Simple Registration to register at the RD that was announced to it, sending this request from its UDP port "[2001:db8:f0::1]:6553":

```
Req: POST coap://[2001:db8:f01::ff]/.well-known/rd?ep=simple-host1

Res: 2.04 Changed
```

Figure 32: Example of a simple registration

The RD would have accepted the registration, and queried the simple host's "/.well-known/core" by itself. As a result, the host is registered as an endpoint in the RD with the name "simple-host1". The registration is active for 90000 seconds, and the endpoint registration Base URI is ""coap://[2001:db8:f0::1]"" following the resolution steps described in Appendix B.1.1. It should be remarked that the Base URI constructed that way always yields a URI of the form: scheme://authority without path suffix.

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching "coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res", obtain "coap://[2001:db8:f0::ff]/rd-lookup/res" as the resource lookup endpoint, and ask it for all temperature resources:

```
Req: GET coap://[2001:db8:f0::ff]/rd-lookup/res?rt=temperature

Res: 2.05 Content
Payload:
<coap://[2001:db8:f0::1]/sensors/temp>;rt=temperature;ct=0
```

Figure 33: Example exchange performing resource lookup

This is not _literally_ the same response that it would have received from a multicast request, but it contains the equivalent statement:

’"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/sensors/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.’

To complete the examples, the client could also query all resources hosted at the endpoint with the known endpoint name "simple-host1":

```
Req: GET coap://[2001:db8:f0::ff]/rd-lookup/res?ep=simple-host1

Res: 2.05 Content
Payload:
<coap://[2001:db8:f0::1]/sensors/temp>;rt=temperature;ct=0,
<coap://[2001:db8:f0::1]/sensors/light>;rt=light-lux;ct=0,
<coap://[2001:db8:f0::1]/t>;
    anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;
    anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=describedby
```

Figure 34: Extended example exchange performing resource lookup

All the target and anchor references are already in absolute form there, which don’t need to be resolved any further.

Had the simple host done an equivalent full registration with a base= parameter (e.g. "?ep=simple-host1&base=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

```
<coap+tcp://simple-host1.example.com/sensors/temp>;rt=temperature;ct=0
```

Figure 35: Example payload of a response to a resource lookup with a dedicated base URI

and analogous records.

B.4. A note on differences between link-format and Link header fields

While link-format and Link header fields look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC8288]. When implementing an RD or interacting with an RD, care must be taken to follow the [RFC6690] behavior whenever application/link-format representations are used.

* "Default value of anchor": Both under [RFC6690] and [RFC8288], relative references in the term inside the angle brackets (the target) and the anchor attribute are resolved against the relevant base URI (which usually is the URI used to retrieve the entity), and independent of each other.

  When, in an [RFC8288] Link header, the anchor attribute is absent, the link's context is the URI of the selected representation (and usually equal to the base URI).

  In [RFC6690] links, if the anchor attribute is absent, the default value is the Origin of (for all relevant cases: the URI reference "/" resolved against) the link's target.

* There is no percent encoding in link-format documents.

  A link-format document is a UTF-8 encoded string of Unicode characters and does not have percent encoding, while Link header fields are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explicitly when required.

  For example, while a Link header field in a page about a Swedish city might read

  Link: </temperature/Malm%C3%B6>;rel=live-environment-data

  a link-format document from the same source might describe the link as

  </temperature/Malmö>;rel=live-environment-data

Appendix C. Limited Link Format

The CoRE Link Format as described in [RFC6690] has been interpreted differently by implementers, and a strict implementation rules out some use cases of an RD (e.g. base values with path components in combination with absent anchors).

This appendix describes a subset of link format documents called Limited Link Format. The one rule herein is not very limiting in practice -- all examples in RFC6690, and all deployments the authors are aware of already stick to them -- but ease the implementation of RD servers.

It is applicable to representations in the application/link-format media type, and any other media types that inherit [RFC6690] Section 2.1.

A link format representation is in Limited Link format if, for each link in it, the following applies:

All URI references either follow the URI or the path-absolute ABNF rule of RFC3986 (i.e. target and anchor each either start with a scheme or with a single slash).

Authors' Addresses

Christian Amsüss (editor)
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: christian@amsuess.com

Zach Shelby
ARM
150 Rose Orchard
San Jose, 95134
United States of America

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View, 94043
United States of America

Phone: +1-707-502-5136
Email: Michael.Koster@smartthings.com

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

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

Peter van der Stok
consultant

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 7, 2016

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

April 5, 2016

Media Types for Sensor Markup Language (SenML)
draft-jennings-core-senml-06

Abstract

This specification defines media types for representing simple sensor measurements and device parameters in the Sensor Markup Language (SenML). Representations are defined in JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR), eXtensible Markup Language (XML), and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use this media type in protocols such as HTTP or CoAP to transport the measurements of the sensor or to be configured.

Status of This Memo

Copyright Notice

Table of Contents

1. Overview

Connecting sensors to the internet is not new, and there have been many protocols designed to facilitate it. This specification defines new media types for carrying simple sensor information in a protocol such as HTTP or CoAP called the Sensor Markup Language (SenML). This format was designed so that processors with very limited capabilities could easily encode a sensor measurement into the media type, while at the same time a server parsing the data could relatively efficiently collect a large number of sensor measurements. The markup language can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and the web resource model where the sensor is requested as a resource representation (e.g., "GET /sensor/temperature").

There are many types of more complex measurements and measurements that this media type would not be suitable for. SenML strikes a balance between having some information about the sensor carried with the sensor data so that the data is self describing but it also tries to make that a fairly minimal set of auxiliary information for efficiency reason. Other information about the sensor can be discovered by other methods such as using the CoRE Link Format [RFC6690].

SenML is defined by a data model for measurements and simple meta-data about measurements and devices. The data is structured as a single array that contains a series of SenML Records which can each contain attributes such as an unique identifier for the sensor, the time the measurement was made, the unit the measurement is in, and the current value of the sensor. Serializations for this data model are defined for JSON [RFC7159], CBOR [RFC7049], XML, and Efficient XML Interchange (EXI) [W3C.REC-exi-20110310].

For example, the following shows a measurement from a temperature gauge encoded in the JSON syntax.

```
[{ "n": "urn:dev:ow:10e2073a01080063", "v":23.1, "u":"Cel" }]
```

In the example above, the array has a single SenML Record with a measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a current value of 23.5 degrees Celsius.

2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets on mesh networks from large numbers of constrained devices. Keeping the total size of payload under 80 bytes makes this easy to use on a wireless mesh network. It is always difficult to

define what small code is, but there is a desire to be able to implement this in roughly 1 KB of flash on a 8 bit microprocessor. Experience with Google power meter and large scale deployments has indicated that the solution needs to support allowing multiple measurements to be batched into a single HTTP or CoAP request. This "batch" upload capability allows the server side to efficiently support a large number of devices. It also conveniently supports batch transfers from proxies and storage devices, even in situations where the sensor itself sends just a single data item at a time. The multiple measurements could be from multiple related sensors or from the same sensor but at different times.

The basic design is an array with a series of measurements. The following example shows two measurements made at different times. The value of a measurement is in the "v" tag, the time of a measurement is in the "t" tag, the "n" tag has a unique sensor name, and the unit of the measurement is carried in the "u" tag.

```
[
    { "n": "urn:dev:ow:10e2073a01080063",
      "t": 1276020076, "v":23.5, "u":"Cel" },
    { "n": "urn:dev:ow:10e2073a01080063",
      "t": 1276020091, "v":23.6, "u":"Cel" }
]
```

To keep the messages small, it does not make sense to repeat the "n" tag in each SenML Record so there is a concept of a Base Name which is simply a string that is prepended to the Name field of all elements in that record and any records that follow it. So a more compact form of the example above is the following.

```
[
    { "bn": "urn:dev:ow:10e2073a01080063",
      "t": 1276020076, "v":23.5, "u":"Cel" },
    {  "t": 1276020091, "v":23.6, "u":"Cel" }
]
```

In the above example the Base Name is in the "bn" tag and the "n" tags in each Record are the empty string so they are omitted. The Base Name also could be put in a separate Record such as in the following example.

```
[
    { "bn": "urn:dev:ow:10e2073a01080063" },
    { "t": 1276020076, "v":23.5, "u":"Cel" },
    { "t": 1276020091, "v":23.6, "u":"Cel" }
]
```

Some devices have accurate time while others do not so SenML supports absolute and relative times. Time is represented in floating point as seconds and values greater than zero represent an absolute time relative to the unix epoch while values of 0 or less represent a relative time in the past from the current time. A simple sensor with no absolute wall clock time might take a measurement every second and batch up 60 of them then send it to a server. It would include the relative time the measurement was made to the time the batch was send in the SenML Pack. The server might have accurate NTP time and use the time it received the data, and the relative offset, to replace the times in the SenML with absolute times before saving the SenML Pack in a document database.

3. Terminology

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

This document also uses the following terms:

SenML Record: One measurement or configuration instance in time presented using the SenML data model.

SenML Pack: One or more SenML Records in an array structure.

4. Semantics

Each SenML Pack carries a single array that represents a set of measurements and/or parameters. This array contains a series of objects with several optional attributes described below:

Base Name: This is a string that is prepended to the names found in the entries. This attribute is optional. This applies to the entries in all Records. A Base Name can only be included in the first Record of the array.

Base Time: A base time that is added to the time found in an entry. This attribute is optional. This applies to the entries in all Records. A Base Time can only be included in the first Record of the array.

Base Unit: A base unit that is assumed for all entries, unless otherwise indicated. This attribute is optional. If a record does not contain a unit value, then the base unit is used otherwise the value of found in the Unit is used. This applies to

the entries in all Records. A Base Unit can only be included in the first object of the array.

Base Value: A base value is added to the value found in an entry, similar to Base Time. This attribute is optional. This applies to the entries in all Records. A Base Value can only be included in the first Record of the array.

Version: Version number of media type format. This attribute is optional positive integer and defaults to 5 if not present. A Version can only be included in the first object of the array.

Name: Name of the sensor or parameter. When appended to the Base Name attribute, this must result in a globally unique identifier for the resource. The name is optional, if the Base Name is present. If the name is missing, Base Name must uniquely identify the resource. This can be used to represent a large array of measurements from the same sensor without having to repeat its identifier on every measurement.

Unit: Units for a measurement value. Optional. If the Record has not Unit, the Base Unit is used as the Unit. Having no Unit and no Base Unit is allowed.

Value Value of the entry. Optional if a Sum value is present, otherwise required. Values are represented using three basic data types, Floating point numbers ("v" field for "Value"), Booleans ("vb" for "Boolean Value"), Strings ("vs" for "String Value") and Data ("vd" for "Binary Data Value") . Exactly one of these three fields MUST appear unless there is Sum field in which case it is allowed to have no Value field or to have "v" field.

Sum: Integrated sum of the values over time. Optional. This attribute is in the units specified in the Unit value multiplied by seconds.

Time: Time when value was recorded. Optional.

Update Time: An optional time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. This can be used to detect the failure of sensors or communications path from the sensor.

The SenML format can be extended with further custom attributes. TODO - describe what extensions are possible and how to do them.

Systems reading one of the objects MUST check for the Version attribute. If this value is a version number larger than the version

which the system understands, the system SHOULD NOT use this object. This allows the version number to indicate that the object contains mandatory to understand attributes. New version numbers can only be defined in an RFC that updates this specification or 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 some bit string that has guaranteed uniqueness (such as a 1-wire address) that is assigned to the device. Some of the examples in this draft use the device URN type as specified in [I-D.arkko-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name. Note that long-term stable unique identifiers are problematic for privacy reasons [RFC7721] and should be used with care or avoided.

The resulting concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", "-", ":", ".", or "_" and it MUST start with a character out of the set "A" to "Z", "a" to "z", or "0" to "9". This restricted character set was chosen so that these names can be directly used as in other types of URI including segments of an HTTP path with no special encoding and can be directly used in many databases and analytic systems. [RFC5952] contains advice on encoding an IPv6 address in a name.

If either the Base Time or Time value is missing, the missing attribute is considered to have a value of zero. The Base Time and Time values are added together to get the time of measurement. A time of zero indicates that the sensor does not know the absolute time and the measurement was made roughly "now". A negative value is used to indicate seconds in the past from roughly "now". A positive value is used to indicate the number of seconds, excluding leap seconds, since the start of the year 1970 in UTC.

Representing the statistical characteristics of measurements, such as accuracy, can be very complex. Future specification may add new attributes to provide better information about the statistical properties of the measurement.

5. Associating Meta-data

SenML is designed to carry the minimum dynamic information about measurements, and for efficiency reasons does not carry significant static meta-data about the device, object or sensors. Instead, it is assumed that this meta-data is carried out of band. For web resources using SenML Packs, this meta-data can be made available using the CoRE Link Format [RFC6690]. The most obvious use of this link format is to describe that a resource is available in a SenML format in the first place. The relevant media type indicator is included in the Content-Type (ct=) attribute.

6. JSON Representation (application/senml+json)

Record atributes:

SenML	JSON	Type
Base Name	bn	String
Base Time	bt	Number
Base Unit	bu	String
Base Value	bv	Number
Version	bver	Number
Name	n	String
Unit	u	String
Value	v	Number
String Value	vs	String
Boolean Value	vb	Boolean
Data Value	vd	String
Value Sum	s	Number
Time	t	Number
Update Time	ut	Number

The root content consists of an array with JSON objects for each SenML Record. All the fields in the above table MAY occur in the records with the type specified in the table.

Only the UTF-8 form of JSON is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC4627]. Characters in the Data Value are base64 encoded with URL safe alphabet as defined in Section 5 of [RFC4648].

Systems receiving measurements MUST be able to process the range of floating point numbers that are representable as an IEEE double-precision floating-point numbers [IEEE.754.1985]. The number of significant digits in any measurement is not relevant, so a reading

of 1.1 has exactly the same semantic meaning as 1.10. If the value has an exponent, the "e" MUST be in lower case. The mantissa SHOULD be less than 19 characters long and the exponent SHOULD be less than 5 characters long. This allows time values to have better than micro second precision over the next 100 years.

6.1. Examples

TODO - simplify examples

TODO - Examples are messed up on if time is an integer or float

TODO - Add example with string, data, boolean, and base value

6.1.1. Single Datapoint

The following shows a temperature reading taken approximately "now" by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

```
[{ "n": "urn:dev:ow:10e2073a01080063", "v":23.1, "u":"Cel" }]
```

6.1.2. Multiple Datapoints

The following example shows voltage and current now, i.e., at an unspecified time.

```
[{"bn": "urn:dev:ow:10e2073a01080063/"},
 { "n": "voltage", "t": 0, "u": "V", "v": 120.1 },
 { "n": "current", "t": 0, "u": "A", "v": 1.2 }
]
```

The next example is similar to the above one, but shows current at Tue Jun 8 18:01:16 UTC 2010 and at each second for the previous 5 seconds.

```
[{"bn": "urn:dev:ow:10e2073a01080063/",
  "bt": 1276020076.001,
  "bu": "A",
  "bver": 5},
   { "n": "voltage", "u": "V", "v": 120.1 },
   { "n": "current", "t": -5, "v": 1.2 },
   { "n": "current", "t": -4, "v": 1.30 },
   { "n": "current", "t": -3, "v": 0.14e1 },
   { "n": "current", "t": -2, "v": 1.5 },
   { "n": "current", "t": -1, "v": 1.6 },
   { "n": "current", "t": 0,  "v": 1.7 }
]
```

Note that in some usage scenarios of SenML the implementations MAY store or transmit SenML in a stream-like fashion, where data is collected over time and continuously added to the object. This mode of operation is optional, but systems or protocols using SenML in this fashion MUST specify that they are doing this. SenML defines a separate mime type (TODO) to indicate Sensor Streaming Markup Language (SensML) for this usage. In this situation the SensML stream can be sent and received in a partial fashion, i.e., a measurement entry can be read as soon as the SenML Record is received and not have to wait for the full SensML Stream to be complete.

For instance, the following stream of measurements may be sent via a long lived HTTP POST from the producer of a SensML to the consumer of that, and each measurement object may be reported at the time it measured:

```
[ {"bn": "urn:dev:ow:10e2073a01080063",
  "bt": 1320067464,
  "bu": "%RH"},
   { "v": 21.2, "t": 0 },
   { "v": 21.3, "t": 10 },
   { "v": 21.4, "t": 20 },
   { "v": 21.4, "t": 30 },
   { "v": 21.5, "t": 40 },
   { "v": 21.5, "t": 50 },
   { "v": 21.5, "t": 60 },
   { "v": 21.6, "t": 70 },
   { "v": 21.7, "t": 80 },
   { "v": 21.5, "t": 90 },
...
```

6.1.3. Multiple Measurements

The following example shows humidity measurements from a mobile device with an IPv6 address 2001:db8::1, starting at Mon Oct 31 13:24:24 UTC 2011. The device also provides position data, which is provided in the same measurement or parameter array as separate entries. Note time is used to for correlating data that belongs together, e.g., a measurement and a parameter associated with it. Finally, the device also reports extra data about its battery status at a separate time.

```
[{"bn": "urn:dev:ow:10e2073a01080063",
  "bt": 1320067464,
  "bu": "%RH"},
   { "v": 20.0, "t": 0 },
   { "v": 24.30621, "u": "lon", "t": 0 },
   { "v": 60.07965, "u": "lat", "t": 0 },
   { "v": 20.3, "t": 60 },
   { "v": 24.30622, "u": "lon", "t": 60 },
   { "v": 60.07965, "u": "lat", "t": 60 },
   { "v": 20.7, "t": 120 },
   { "v": 24.30623, "u": "lon", "t": 120 },
   { "v": 60.07966, "u": "lat", "t": 120 },
   { "v": 98.0, "u": "%EL", "t": 150 },
   { "v": 21.2, "t": 180 },
   { "v": 24.30628, "u": "lon", "t": 180 },
   { "v": 60.07967, "u": "lat", "t": 180 }
]
```

The size of this example represented in various forms, as well as that form compressed with gzip is given in the following table.

Encoding	Size	Compressed Size
JSON	567	200
XML	656	232
CBOR	292	192
EXI	160	183

Table 1: Size Comparisons

Note the CBOR and EXI sizes are not using the schema guidance so the could be a bit smaller.

6.1.4. Collection of Resources

The following example shows how to query one device that can provide multiple measurements. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on http://[2001:db8::2] at Mon Oct 31 16:27:09 UTC 2011, and has gotten two separate values as a result, a temperature and humidity measurement.

```
[{"bn": "urn:dev:ow:10e2073a01080063/",
  "bt": 1320078429,
  "bver": 5
  },
  { "n": "temperature", "v": 27.2, "u": "Cel" },
  { "n": "humidity", "v": 80, "u": "%RH" }
]
```

7. CBOR Representation (application/senml+cbor)

The CBOR [RFC7049] representation is equivalent to the JSON representation, with the following changes:

- o For compactness, the CBOR representation uses integers for the map keys defined in Table 2. This table is conclusive, i.e., there is no intention to define any additional integer map keys; any extensions will use string map keys.

- o For JSON Numbers, the CBOR representation can use integers, floating point numbers, or decimal fractions (CBOR Tag 4); the common limitations of JSON implementations are not relevant for these. For the version number, however, only an unsigned integer is allowed.

Name	JSON label	CBOR label
Version	bver	-1
Base Name	bn	-2
Base Time	bt	-3
Base Units	bu	-4
Base Value	bv	-5
Name	n	0
Units	u	1
Value	v	2
String Value	vs	3
Boolean Value	vb	4
Value Sum	s	5
Time	t	6
Update Time	ut	7
Data Value	vd	8

Table 2: CBOR representation: integers for map keys

The following example shows an hexdump of the CBOR example for the same sensor measurement as in Section 6.1.2.

```
0000 88 a4 62 62 6e 78 1c 75 72 6e 3a 64 65 76 3a 6f |..bbnx.urn:dev:o|
0010 77 3a 31 30 65 32 30 37 33 61 30 31 30 38 30 30 |w:10e2073a010800|
0020 36 33 2f 62 62 74 1a 4c 0e 85 6c 62 62 75 61 41 |63/bbt.L..lbbuaA|
0030 63 76 65 72 05 a3 61 6e 67 76 6f 6c 74 61 67 65 |cver..angvoltage|
0040 61 75 61 56 61 76 fb 40 5e 06 66 66 66 66 66 a3 |auaVav.@^.fffff.|
0050 61 6e 67 63 75 72 72 65 6e 74 61 74 24 61 76 fb |angcurrentat$av.|
0060 3f f3 33 33 33 33 33 33 a3 61 6e 67 63 75 72 72 |?.333333.angcurr|
0070 65 6e 74 61 74 23 61 76 fb 3f f4 cc cc cc cc cc |entat#av.?......|
0080 cd a3 61 6e 67 63 75 72 72 65 6e 74 61 74 22 61 |..angcurrentat"a|
0090 76 fb 3f f6 66 66 66 66 66 66 a3 61 6e 67 63 75 |v.?.ffffff.angcu|
00a0 72 72 65 6e 74 61 74 21 61 76 fb 3f f8 00 00 00 |rrentat!av.?....|
00b0 00 00 00 a3 61 6e 67 63 75 72 72 65 6e 74 61 74 |....angcurrentat|
00c0 20 61 76 fb 3f f9 99 99 99 99 99 9a a2 61 6e 67 | av.?........ang|
00d0 63 75 72 72 65 6e 74 61 76 fb 3f fb 33 33 33 33 |currentav.?.3333|
00e0 33 33 0a                                        |33.|
00e3
```

8. XML Representation (application/senml+xml)

A SenML Stream can also be represented in XML format as defined in this section. The following example shows an XML example for the same sensor measurement as in Section 6.1.2.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a01080063/" bt="1.276020076e+09"
  bu="A" bver="5"></senml>
  <senml n="voltage" u="V" v="120.1"></senml>
  <senml n="current" t="-5" v="1.2"></senml>
  <senml n="current" t="-4" v="1.3"></senml>
  <senml n="current" t="-3" v="1.4"></senml>
  <senml n="current" t="-2" v="1.5"></senml>
  <senml n="current" t="-1" v="1.6"></senml>
  <senml n="current" v="1.7"></senml>
</sensml>
```

The SenML Stream is represented as a sensml tag that contains a series of senml tags for each SenML Record. The SenML Fields are represents as XML attributes. The following table shows the mapping the SenML Field names to the attribute used in the XML senml tag.

SenML Field	XML	Type
Base Name	bn	string
Base Time	bt	float
Base Unit	bu	string
Base Value	bv	float
Version	bver	int
Name	n	string
Unit	u	string
Value	v	float
String Value	vs	string
Data Value	vd	string
Boolean Value	vb	boolean
Value Sum	s	float
Time	t	float
Update Time	ut	float

The RelaxNG schema for the XML is:

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

link = element l {
      attribute * { xsd:string }*
}


senml = element senml {
  attribute bn { xsd:string }?,
  attribute bt { xsd:double }?,
  attribute bv { xsd:double }?,
  attribute bu { xsd:string }?,
  attribute bver { xsd:int }?,

  attribute n { xsd:string }?,
  attribute s { xsd:double }?,
  attribute t { xsd:double }?,
  attribute u { xsd:string }?,
  attribute ut { xsd:double }?,

  attribute v { xsd:double }?,
  attribute vb { xsd:boolean }?,
  attribute vs { xsd:string }?,
  attribute vd { xsd:string }?,

  link*
}

sensml =
  element sensml {
    senml+
}

start = sensml
```

9. EXI Representation (application/senml-exi)

For efficient transmission of SenML over e.g. a constrained network, Efficient XML Interchange (EXI) can be used. This encodes the XML Schema structure of SenML into binary tags and values rather than ASCII text. An EXI representation of SenML SHOULD be made using the strict schema-mode of EXI. This mode however does not allow tag extensions to the schema, and therefore any extensions will be lost in the encoding. For uses where extensions need to be preserved in EXI, the non-strict schema mode of EXI MAY be used.

The EXI header option MUST be included. An EXI schemaID options MUST be set to the value of "a" indicating the scheme provided in this specification. Future revisions to the schema can change this schemaID to allow for backwards compatibility. When the data will be transported over CoAP or HTTP, an EXI Cookie SHOULD NOT be used as it simply makes things larger and is redundant to information provided in the Content-Type header.

TODO - examples probably have the wrong setting the schemaID

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

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="urn:ietf:params:xml:ns:senml"
xmlns:ns1="urn:ietf:params:xml:ns:senml">
  <xs:element name="l">
    <xs:complexType>
      <xs:anyAttribute processContents="skip" />
    </xs:complexType>
  </xs:element>
  <xs:element name="senml">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded"
        ref="ns1:l" />
      </xs:sequence>
      <xs:attribute name="bn" type="xs:string" />
      <xs:attribute name="bt" type="xs:double" />
      <xs:attribute name="bv" type="xs:double" />
      <xs:attribute name="bu" type="xs:string" />
      <xs:attribute name="bver" type="xs:int" />
      <xs:attribute name="n" type="xs:string" />
      <xs:attribute name="s" type="xs:double" />
      <xs:attribute name="t" type="xs:double" />
      <xs:attribute name="u" type="xs:string" />
      <xs:attribute name="ut" type="xs:double" />
      <xs:attribute name="v" type="xs:double" />
      <xs:attribute name="vb" type="xs:boolean" />
      <xs:attribute name="vs" type="xs:string" />
      <xs:attribute name="vd" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="sensml">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="ns1:senml" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The following shows a hexdump of the EXI produced from encoding the following XML example. Note this example is the same information as the first example in Section 6.1.2 in JSON format.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a01080063/"></senml>
  <senml n="voltage" u="V" v="120.1"></senml>
  <senml n="current" u="A" v="1.2"></senml>
</sensml>
```

Which compresses with EXI to the following displayed in hexdump:

```
0000 a0 30 41 cd 95 b9 b5 b0 d4 b9 9d 95 b8 b9 e1 cd |.0A.............|
0010 91 00 f3 ab 93 71 d3 23 2b b1 d3 7b b9 d1 89 83 |.....q.#+..{....|
0020 29 91 81 b9 9b 09 81 89 81 c1 81 81 b1 99 7f 14 |)...............|
0030 25 d9 bd b1 d1 85 9d 94 80 d5 8a c4 26 01 0a 12 |%...........&...|
0040 c6 ea e4 e4 ca dc e8 40 68 24 19 00 90          |.......@h$...|
004d
```

The above example used the bit packed form of EXI but it is also possible to use a byte packed form of EXI which can makes it easier for a simple sensor to produce valid EXI without really implementing EXI. Consider the example of a temperature sensor that produces a value in tenths of degrees Celsius over a range of 0.0 to 55.0. It would produce an XML SenML file such as:

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml n="urn:dev:ow:10e2073a01080063" u="Cel" v="23.1"></senml>
</sensml>
```

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

```
0000 a0 00 48 82 0e 6c ad cd ad 86 a5 cc ec ad c5 cf |..H..l..........|
0010 0e 6c 80 02 05 1d 75 72 6e 3a 64 65 76 3a 6f 77 |.l....urn:dev:ow|
0020 3a 31 30 65 32 30 37 33 61 30 31 30 38 30 30 36 |:10e2073a0108006|
0030 33 02 05 43 65 6c 01 00 e7 01 01 00 04 01       |3..Cel........|
003e
```

A small temperature sensor devices that only generates this one EXI file does not really need an full EXI implementation. It can simple hard code the output replacing the one wire device ID starting at byte 0x16 and going to byte 0x31 with it’s device ID, and replacing the value "0xe7 0x01" at location 0x38 to 0x39 with the current temperature. The EXI Specification [W3C.REC-exi-20110310] contains the full information ’on how floating point numbers are represented, but for the purpose of this sensor, the temperature can be converted to an integer in tenths of degrees (231 in this example). EXI stores 7 bits of the integer in each byte with the top bit set to one if there are further bytes. So the first bytes at is set to low 7 bits of the integer temperature in tenths of degrees plus 0x80. In this example 231 & 0x7F + 0x80 = 0xE7. The second byte is set to the

integer temperature in tenths of degrees right shifted 7 bits. In this example 231 >> 7 = 0x01.

10. Usage Considerations

The measurements support sending both the current value of a sensor as well as the an integrated sum. For many types of measurements, the sum is more useful than the current value. For example, an electrical meter that measures the energy a given computer uses will typically want to measure the cumulative amount of energy used. This is less prone to error than reporting the power each second and trying to have something on the server side sum together all the power measurements. If the network between the sensor and the meter goes down over some period of time, when it comes back up, the cumulative sum helps reflect what happened while the network was down. A meter like this would typically report a measurement with the units set to watts, but it would put the sum of energy used in the "s" attribute of the measurement. It might optionally include the current power in the "v" attribute.

While the benefit of using the integrated sum is fairly clear for measurements like power and energy, it is less obvious for something like temperature. Reporting the sum of the temperature makes it easy to compute averages even when the individual temperature values are not reported frequently enough to compute accurate averages. Implementors are encouraged to report the cumulative sum as well as the raw value of a given sensor.

Applications that use the cumulative sum values need to understand they are very loosely defined by this specification, and depending on the particular sensor implementation may behave in unexpected ways. Applications should be able to deal with the following issues:

1. Many sensors will allow the cumulative sums to "wrap" back to zero after the value gets sufficiently large.

2. Some sensors will reset the cumulative sum back to zero when the device is reset, loses power, or is replaced with a different sensor.

3. Applications cannot make assumptions about when the device started accumulating values into the sum.

Typically applications can make some assumptions about specific sensors that will allow them to deal with these problems. A common assumption is that for sensors whose measurement values are always positive, the sum should never get smaller; so if the sum does get

smaller, the application will know that one of the situations listed above has happened.

11. CDDL

For reference, the CBOR representation can be described with the CDDL [I-D.greevenbosch-appsawg-cbor-cddl] specification in Figure 1.

```
SenML-Pack = [initial-record, * follow-on-record]

initial-record = initial-defined .and initial-generic
follow-on-record = follow-on-defined .and follow-on-generic

; first do a specification of the labels as defined:

initial-defined = {
  ? bn => tstr,        ; Base Name
  ? bt => numeric,     ; Base Time
  ? bu => tstr,        ; Base Units
  ? bv => numeric,     ; Base value
  ? bver => uint,      ; Base Version
  follow-on-defined-group,
  + base-key-value-pair
}

follow-on-defined-group = (
   ? n => tstr,        ; Name
   ? u => tstr,        ; Units
   ? ( v => numeric // ; Numeric Value
       vs => tstr //   ; String Value
       vb => bool //   ; Boolean Value
       vd => bstr )    ; Data Value
   ? s => numeric,     ; Value Sum
   ? t => numeric,     ; Time
   ? ut => numeric,    ; Update Time
   * key-value-pair
)
follow-on-defined = { follow-on-defined-group }

; CBOR version (use the labels)
bver = -1  n  = 0   s  = 5
bn  = -2   u  = 1   t  = 6
bt  = -3   v  = 2   ut = 7
bu  = -4   vs = 3   vd = 8
bv  = -5   vb = 4
; use the label *names* for JSON

; now define the generic versions
```

```
initial-generic = {
  follow-on-generic-group,
  * base-key-value-pair,
}

follow-on-generic-group = (
  + key-value-pair,
)
follow-on-generic = { follow-on-generic-group }

key-value-pair = ( non-b-label => value )

base-key-value-pair = ( b-label => value )

non-b-label = tstr .regexp  "[A-Zac-z0-9][-_:.A-Za-z0-9]*" / uint
b-label = tstr .regexp  "b[-_:.A-Za-z0-9]+" / nint

value = tstr / bstr / numeric / bool
numeric = number / decfrac
```

Figure 1: CDDL specification for CBOR SenML

12. IANA Considerations

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

12.1. Units Registry

IANA will create a registry of SenML unit symbols. The primary purpose of this registry is to make sure that symbols uniquely map to give type of measurement. Definitions for many of these units can be found in location such as [NIST811] and [BIPM].

Symbol	Description	Type	Reference
m	meter	float	RFC-AAAA
g	gram	float	RFC-AAAA
s	second	float	RFC-AAAA
A	ampere	float	RFC-AAAA
K	kelvin	float	RFC-AAAA
cd	candela	float	RFC-AAAA
mol	mole	float	RFC-AAAA
Hz	hertz	float	RFC-AAAA
rad	radian	float	RFC-AAAA
sr	steradian	float	RFC-AAAA

N	newton	float	RFC-AAAA
Pa	pascal	float	RFC-AAAA
J	joule	float	RFC-AAAA
W	watt	float	RFC-AAAA
C	coulomb	float	RFC-AAAA
V	volt	float	RFC-AAAA
F	farad	float	RFC-AAAA
Ohm	ohm	float	RFC-AAAA
S	siemens	float	RFC-AAAA
Wb	weber	float	RFC-AAAA
T	tesla	float	RFC-AAAA
H	henry	float	RFC-AAAA
Cel	degrees Celsius	float	RFC-AAAA
lm	lumen	float	RFC-AAAA
lx	lux	float	RFC-AAAA
Bq	becquerel	float	RFC-AAAA
Gy	gray	float	RFC-AAAA
Sv	sievert	float	RFC-AAAA
kat	katal	float	RFC-AAAA
pH	pH acidity	float	RFC-AAAA
%	Value of a switch (note 1)	float	RFC-AAAA
count	counter value	float	RFC-AAAA
%RH	Relative Humidity	float	RFC-AAAA
m2	area	float	RFC-AAAA
l	volume in liters	float	RFC-AAAA
m/s	velocity	float	RFC-AAAA
m/s2	acceleration	float	RFC-AAAA
l/s	flow rate in liters per second	float	RFC-AAAA
W/m2	irradiance	float	RFC-AAAA
cd/m2	luminance	float	RFC-AAAA
Bspl	bel sound pressure level	float	RFC-AAAA
bit/s	bits per second	float	RFC-AAAA
lat	degrees latitude (note 2)	float	RFC-AAAA
lon	degrees longitude (note 2)	float	RFC-AAAA
%EL	remaining battery energy level in percents	float	RFC-AAAA
EL	remaining battery energy level in seconds	float	RFC-AAAA
beat/m	Heart rate in beats per minute	float	RFC-AAAA
beats	Cumulative number of heart beats	float	RFC-AAAA

Table 3

- Note 1: A value of 0.0 indicates the switch is off while 1.0 indicates on and 0.5 would be half on.

- o Note 2: Assumed to be in WGS84 unless another reference frame is known for the sensor.

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider the following guidelines:

1. There needs to be a real and compelling use for any new unit to be added.

2. Units should define the semantic information and be chosen carefully. Implementors need to remember that the same word may be used in different real-life contexts. For example, degrees when measuring latitude have no semantic relation to degrees when measuring temperature; thus two different units are needed.

3. These measurements are produced by computers for consumption by computers. The principle is that conversion has to be easily be done when both reading and writing the media type. The value of a single canonical representation outweighs the convenience of easy human representations or loss of precision in a conversion.

4. Use of SI prefixes such as "k" before the unit is not allowed. Instead one can represent the value using scientific notation such a 1.2e3. TODO - Open Issue. Some people would like to have SI prefixes to improve human readability.

5. For a given type of measurement, there will only be one unit type defined. So for length, meters are defined and other lengths such as mile, foot, light year are not allowed. For most cases, the SI unit is preferred.

6. Symbol names that could be easily confused with existing common units or units combined with prefixes should be avoided. For example, selecting a unit name of "mph" to indicate something that had nothing to do with velocity would be a bad choice, as "mph" is commonly used to mean miles per hour.

7. The following should not be used because the are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, n, u, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi.

8. The following units should not be used as they are commonly used to represent other measurements Ky, Gal, dyn, etg, P, St, Mx, G, Oe, Gb, sb, Lmb, ph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, kWh.

9. The unit names are case sensitive and the correct case needs to be used, but symbols that differ only in case should not be allocated.

10. A number after a unit typically indicates the previous unit raised to that power, and the / indicates that the units that follow are the reciprocal. A unit should have only one / in the name.

11. A good list of common units can be found in the Unified Code for Units of Measure [UCUM].

12.2. SenML label registry

IANA will create a registry for SenML labels. The initial content of the registry are shown in TODO.

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider that shorter labels should have more strict review.

12.3. Media Type Registration

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

12.3.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json and sensml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC7159]. See RFC-AAAA for details. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and

instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any JSON key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "ver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.2. senml+cbor Media Type Registration

Type name: application

Subtype name: senml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.3. senml+xml Media Type Registration

Type name: application

Subtype name: senml+xml and sensml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.4. senml-exi Media Type Registration

Type name: application

Subtype name: senml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.4. XML Namespace Registration

This document registers the following XML namespaces in the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:senml

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces

12.5. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the SenML media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. All IDs are assigned from the "Expert Review" (0-255) range. The assigned IDs are show in Table 4.

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

Table 4: CoAP Content-Format IDs

13. Security Considerations

See Section 14. Further discussion of security properties can be found in Section 12.3.

14. Privacy Considerations

Sensor data can range from information with almost no security considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transport protocol such as S/MIME or HTTP with TLS that can provide integrity, confidentiality, and authentication information about the source of the data.

15. Acknowledgement

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

The CBOR Representation text and CDDL was contributed by Carsten Bormann.

16. References

16.1. Normative References

[BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI)", 8th edition, 2006.

[IEEE.754.1985] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE Standard 754, August 1985.

[NIST811] Thompson, A. and B. Taylor, "Guide for the Use of the International System of Units (SI)", NIST Special Publication 811, 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <http://www.rfc-editor.org/info/rfc4627>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

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

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <http://www.rfc-editor.org/info/rfc7303>.

[W3C.REC-exi-20110310] Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <http://www.w3.org/TR/2011/REC-exi-20110310>.

16.2. Informative References

[I-D.arkko-core-dev-urn] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-03 (work in progress), July 2012.

[I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-08 (work in progress), March 2016.

[I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-04 (work in progress), November 2015.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, DOI 10.17487/RFC2141, May 1997, <http://www.rfc-editor.org/info/rfc2141>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <http://www.rfc-editor.org/info/rfc4122>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <http://www.rfc-editor.org/info/rfc7721>.

[UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics, 2013, <http://unitsofmeasure.org/ucum.html>.

Appendix A. Links extension

An extension to SenML to support links is expected to be registered and defined by [I-D.ietf-core-links-json].

The link extension can be an array of objects that can be used for additional information. Each object in the Link array is constrained to being a map of strings to strings with unique keys.

The following shows an example of the links extension.

```
[{"bn": "urn:dev:ow:10e2073a01080063/",
  "bt": 1320078429,
  "l": "[{\"href\":\"humidity\",\"foo\":\"bar1\"}\"
  },
  { "n": "temperature", "v": 27.2, "u": "Cel" },
  { "n": "humidity", "v": 80, "u": "%RH" }
]
```

Authors' Addresses

Cullen Jennings
Cisco
400 3rd Avenue SW
Calgary, AB T2P 4H2
Canada

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2017

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson

July 7, 2016

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-koster-core-coap-pubsub-05

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports machine-to-machine communication across networks of constrained devices. CoAP uses a request/response model where clients make requests to servers in order to request actions on resources. Depending on the situation the same device may act either as a server or a client.

One important class of constrained devices includes devices that are intended to run for years from a small battery, or by scavenging energy from their environment. These devices have limited

reachability because they spend most of their time in a sleeping state with no network connectivity. Devices may also have limited reachability due to certain middle-boxes, such as Network Address Translators (NATs) or firewalls. Such middle-boxes often prevent connecting to a device from the Internet unless the connection was initiated by the device.

This document specifies the means for nodes with limited reachability to communicate using simple extensions to CoAP. The extensions enable publish-subscribe communication using a broker node that enables store-and-forward messaging between two or more nodes. Furthermore the extensions facilitate many-to-many communication using CoAP.

2. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format [RFC6570] is used to describe the REST interfaces defined in this specification.

This specification makes use of the following additional terminology:

Publish-Subscribe (pubsub): A messaging paradigm where messages are published to a broker and potential receivers can subscribe to the broker to receive messages. The publishers do not (need to) know where the message will be eventually sent: the publications and subscriptions are matched by a broker and publications are delivered by the broker to subscribed receivers.

CoAP pubsub function set: A group of well-known REST resources that together provide the CoAP pubsub service.

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

CoAP pubsub Client: A CoAP client that implements the CoAP pubsub function set.

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

3. Architecture

3.1. CoAP pubsub Architecture

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

```
Clients        pubsub         Broker
+-------+         |
| CoAP  |         |
|pubsub |---------|------+
|Client |         |      |    +-------+
+-------+         |      +----| CoAP  |
                  |           |pubsub |
+-------+         |      +----|Broker |
| CoAP  |         |      |    +-------+
|pubsub |---------|------+
|Client |         |
+-------+         |
```

Figure 1: CoAP pubsub Architecture

3.2. CoAP pubsub Broker

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

3.3. CoAP pubsub Client

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

3.4. CoAP pubsub Topic

The clients and broker use topics to identify a particular resource or object in a publish-subscribe system. Topics are conventionally formed as a hierarchy, e.g. "/sensors/weather/barometer/pressure" or "EP-33543/sen/3303/0/5700". The topics are hosted at the broker and all the clients using the broker share the same namespace for topics. A CoAP pubsub topic has a reference path using URI path [RFC3986] construction, link attributes [RFC6690], and a representation of a value with specified content-formats. A CoAP pubsub topic value may alternatively be a collection of one or more sub-topics, consisting of links to the sub-topic URIs and indicated by a link-format content-format.

3.5. Brokerless pubsub

Figure 2 shows an arrangement for using CoAP pubsub in a "brokerless" configuration between peer nodes. Nodes in a brokerless system act as both broker and client. The Broker interface in a brokerless node may be pre-configured with topics that expose services and resources. Brokerless peer nodes can be mixed with client and broker nodes in a system with full interoperability.

```
  Peer         pubsub          Peer
+-------+         |         +-------+
| CoAP  |         |         | CoAP  |
|pubsub |---------|---------|pubsub |
|Client |         |         |Broker |
+-------+         |         +-------+
| CoAP  |         |         | CoAP  |
|pubsub |---------|---------|pubsub |
|Broker |         |         |Client |
+-------+         |         +-------+
```

Figure 2: Brokerless pubsub

4. CoAP pubsub Function Set

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

4.1. DISCOVER

CoAP pubsub Clients discover CoAP pubsub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [I-D.ietf-core-resource-directory]. A CoAP pubsub Broker SHOULD indicate its presence and availability on a network by exposing a link to its pubsub function set at its .well-known/core location [RFC6690]. A CoAP pubsub broker MAY register its pubsub function set location with a Resource Directory. Figure 3 shows an example of a client discovering a local pubsub Function Set using CoAP Simple Discovery. A broker wishing to advertise the CoAP pubsub Function Set for Simple Discovery or through a Resource Directory MUST use the link relation rt="core.ps". A broker MAY advertise it’s supported content formats and other attributes in the link to it’s pubsub function set.

A CoAP pubsub Broker MAY offer the Discover interface to enable Clients to find topics of interest, either by topic name or by link attributes which may be registered when the topic is created. Figure 4 shows an example of a client looking for a topic with a resource type (rt) of "temperature" in the pubsub function set /ps using the Discover interface. The client then receives the URI of the resource and its content-format.

A CoAP pubsub Broker MAY expose the Discover interface through the .well-known/core resource. Links to topics may be exposed at .well-known/core in addition to links to the pubsub function set. Figure 5 shows an example of topic discovery through .well-known/core.

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: /.well-known/core

URI Template: /{+ps/}{topic}{/topic*}{?q*}

URI Template Variables:

/.well-known/core := for discovering the pubsub function set (optional)

ps := pubsub Function Set path (optional). The path of the pubsub Function Set, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: application/link-format

The following response codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the broker resource. A pubsub broker SHOULD use the value "/ps/" for the function set URI wherever possible.

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

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

```
Client                                          Broker
  |                                               |
  | ------ GET /.well-known/core?rt=core.ps ----->|
  | ---Content-Format: application/link-format----|
  |                                               |
  | <--2.05 Content "</ps/>;rt="core.ps";ct=40----|
  |                                               |
```

Figure 3: Example of DISCOVER pubsub function

```
Client                                          Broker
   |                                              |
   | ---------- GET /ps/?rt="temperature" -------->|
   |    Content-Format: application/link-format    |
   |                                              |
   | <---2.05 Content                              |
   |   </ps/currentTemp>;rt="temperature";ct=50 ---|
   |                                              |
```

Figure 4: Example of DISCOVER topic

```
Client                                          Broker
   |                                              |
   | -------- GET /.well-known/core?ct=50 -------->|
   |    Content-Format: application/link-format    |
   |                                              |
   | <---2.05 Content                              |
   |   </ps/currentTemp>;rt="temperature";ct=50 ---|
   |                                              |
```

Figure 5: Example of DISCOVER topic

4.2. CREATE

Clients create topics on the broker using the CREATE interface. A client wishing to create a topic MUST use CoAP POST to the pubsub function set location with a payload indicating the desired topic. The topic specification sent in the payload MUST use a supported serialization of the CoRE link format [RFC6690]. The target of the link MUST be a URI formatted string. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Format) link attribute in the link-format payload. The client MAY indicate the lifetime of the topic by including the Max-Age option in the CREATE request. Broker MUST return a response code of "2.01 Created" if the topic is created and return the created relative URI path via Location-Path options. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if a new topic can not be created. Broker SHOULD remove topics if the Max-Age of the topic is exceeded without any publishes to the topic. Broker SHOULD retain a topic indefinitely if the Max-Age option is elided or is set to zero upon topic creation. The lifetime of a topic MUST be refreshed upon create operations with a target of an existing topic.

Topics may be created as sub-topics of other topics. A client MAY create a topic with a ct (Content Format) link attribute value which describes a supported serialization of the CoRE link format [RFC6690] such as application/link-format (ct=40) or its JSON or CBOR serializations. If a topic is created which describes a link serialization, that topic may then have sub-topics created under it as shown in Figure 7.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

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

URI Template Variables:

ps := pubsub Function Set path (mandatory). The path of the pubsub Function Set, as obtained from discovery. A pubsub broker SHOULD use the value "ps" for this variable whenever possible.

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for this interface:

Success: 2.01 "Created". Successful Creation of the topic

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.03 "Forbidden". Topic already exists.

Failure: 4.06 "Not Acceptable". Unsupported content format for topic.

Figure 6 shows an example of a topic called "topic1" being successfully created.

```
Client                                          Broker
   |                                              |
   | ---------- POST /ps "<topic1>;ct=50" -------->|
   |                                              |
   | <---------------- 2.01 Created ---------------|
   |              Location: /ps/topic1            |
   |                                              |
```

Figure 6: Example of CREATE topic

```
Client                                          Broker
   |                                              |
   | ------- POST /ps/ "<mainTopic>;ct=40" ------->|
   |                                              |
   | <---------------- 2.01 Created ---------------|
   |            Location: /ps/mainTopic/          |
   |                                              |
   | --- POST /ps/mainTopic/ "<subTopic>;ct=50" -->|
   |                                              |
   | <---------------- 2.01 Created ---------------|
   |       Location: /ps/mainTopic/subTopic       |
   |                                              |
   |                                              |
```

Figure 7: Example of CREATE sub-topic

4.3. PUBLISH

A CoAP pubsub Client uses the PUBLISH interface for updating topics on the broker. The client MUST use the PUT method to publish state updates to the CoAP pubsub Broker. A client MUST use the content format specified upon creation of a given topic to publish updates to that topic. The broker MUST reject publish operations which do not use the specified content format. A CoAP client publishing on a topic MAY indicate the maximum lifetime of the value by including the Max-Age option in the publish request. The broker MUST return a response code of "2.04 Changed" if the publish is accepted or "4.04 Not Found" if the topic does not exist. A broker MAY return "4.29 Too Many Requests" if simple flow control as described in Section 7 is implemented.

The Broker MUST notify all clients subscribed on a particular topic each time it receives a publish on that topic. An example is shown in Figure 9. If a client publishes to a broker with the Max-Age option, the broker MUST include the same value for the Max-Age option

in all notifications. A broker MUST use CoAP Notification as described in [RFC7641] to notify subscribed clients.

The PUBLISH interface is specified as follows:

Interaction: Client -> Broker

Method: PUT

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

URI Template Variables:

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

topic := The desired topic to publish on.

Content-Format: Any valid CoAP content format

Payload: Representation of the topic value (CoAP resource state representation) in the indicated content format

The following response codes are defined for this interface:

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

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 7).

Figure 8 shows an example of a new value being successfully published to the topic "topic1". See Figure 9 for an example of a broker forwarding a message from a publishing client to a subscribed client.

```
          Client                                          Broker
            |                                               |
            | ---------- PUT /ps/topic1 "1033.3"  --------> |
            |                                               |
            |                                               |
            | <--------------- 2.04 Changed---------------- |
            |                                               |
```

Figure 8: Example of PUBLISH

4.4. SUBSCRIBE

CoAP pubsub Clients subscribe to topics on the Broker using CoAP Observe as described in [RFC7641]. A CoAP pubsub Client wishing to Subscribe to a topic on a broker MUST use a CoAP GET with Observe registration. The Broker MAY add the client to a list of observers. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. The broker MUST accept Subscribe requests on a topic if the content format of the request matches the content format the topic was created with. The broker MAY accept Subscribe requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if it can not return the requested content format. If a Broker is unable to accept a new Subscription on a topic, it SHOULD return the appropriate response code without the Observe option as per as per [RFC7641] Section 4.1. There is no explicit maximum lifetime of a Subscription, thus a Broker may remove subscribers at any time. The Broker, upon removing a Subscriber, will transmit the appropriate response code without the Observe option, as per [RFC7641] Section 4.2, to the removed Subscriber.

The SUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

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

URI Template Variables:

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

topic := The desired topic to subscribe to.

The following response codes are defined for this interface:

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

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

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.15 "Unsupported Content Format". Unsupported content format.

Figure 9 shows an example of Client2 subscribing to "topic1" and receiving a response from the broker, with a subsequent notification. The subscribe response from the broker uses the last stored value associated with the topic1. The notification from the broker is sent in response to the publish received from Client1.

```
  Client1   Client2                                        Broker
    |          |                   Subscribe                 |
    |          | ----- GET /ps/topic1 Observe:0 Token:XX ----> |
    |          |                                             |
    |          | <---------- 2.05 Content Observe:10---------- |
    |          |                                             |
    |          |                                             |
    |          |                    Publish                  |
    | ---------|----------- PUT /ps/topic1 "1033.3"  --------> |
    |          |                    Notify                   |
    |          | <---------- 2.05 Content Observe:11---------- |
    |          |                                             |
```

Figure 9: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

CoAP pubsub Clients unsubscribe from topics on the Broker using the CoAP Cancel Observation operation. A CoAP pubsub Client wishing to unsubscribe to a topic on a Broker MUST either use CoAP GET with Observe using an Observe parameter of 1 or send a CoAP Reset message in response to a publish, as per [RFC7641].

The UNSUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

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

URI Template Variables:

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

topic := The desired topic to unsubscribe from.

The following response codes are defined for this interface:

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

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

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Figure 10 shows an example of a client unsubscribe using the Observe=1 cancellation method.

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

Figure 10: Example of UNSUBSCRIBE

4.6. READ

A CoAP pubsub client wishing to obtain only the most recent published value on a topic MAY use the READ interface. For reading, the client uses the CoAP GET method. The broker MUST accept Read requests on a topic if the content format of the request matches the content format the topic was created with. The broker MAY accept Read requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the topic to be valid since the last publish. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if the broker can not return the requested content format.

The READ interface is specified as follows:

Interaction: Client -> Broker

Method: GET

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

URI Template Variables:

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

topic := The desired topic to READ.

The following response codes are defined for this interface:

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

Success: 2.04 "No Content". Topic exists, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.15 "Unsupported Content Format". Unsupported content-format.

Figure 11 shows an example of a successful READ from topic1, followed by a Publish on the topic, followed at some time later by a read of the updated value from the recent Publish.

```
  Client1   Client2                                        Broker
     |          |                     Read                      |
     |          | --------------- GET /ps/topic1 -------------> |
     |          |                                               |
     |          | <---------- 2.05 Content "1007.1"------------ |
     |          |                                               |
     |          |                                               |
     |          |                    Publish                    |
     | ---------|----------- PUT /ps/topic1 "1033.3"  --------> |
     |          |                                               |
     |          |                                               |
     |          |                     Read                      |
     |          | --------------- GET /ps/topic1 -------------> |
     |          |                                               |
     |          | <----------- 2.05 Content "1033.3"----------- |
     |          |                                               |
```

Figure 11: Example of READ

4.7. REMOVE

A CoAP pubsub Client wishing to remove a topic MAY use the CoAP Delete operation on the URI of the topic. The CoAP pubsub Broker MUST return "2.02 Deleted" if the remove operation is successful. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if the topic can not be removed. When a topic is removed for any reason, the Broker SHOULD return the response code 4.04 Not Found and remove all of the observers from the list of observers as per as per [RFC7641] Section 3.2.

The REMOVE interface is specified as follows:

Interaction: Client -> Broker

Method: DELETE

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

URI Template Variables:

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

topic := The desired topic to REMOVE.

Content-Format: None

Response Payload: None

The following response codes are defined for this interface:

Success: 2.02 "Deleted". Successful remove

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Figure 12 shows a successful remove of topic1.

```
Client                                          Broker
  |                                               |
  | ------------- DELETE /ps/topic1 ------------> |
  |                                               |
  |                                               |
  | <-------------- 2.02 Deleted ---------------- |
  |                                               |
```

Figure 12: Example of REMOVE

5. CoAP pubsub Operation with Resource Directory

A CoAP pubsub Broker may register a pubsub Function Set with a Resource Directory. A pubsub Client may use an RD to discover a pubsub Broker.

A CoAP pubsub Client may register links [RFC6690] with a Resource Directory to enable discovery of created pubsub topics. A pubsub Client may use an RD to discover pubsub Topics. A client which registers pubsub Topics with an RD MUST use the context relation (con) [I-D.ietf-core-resource-directory] to indicate that the context of the registered links is the pubsub Broker.

A CoAP pubsub Broker may alternatively register links to its topics to a Resource Directory by triggering the RD to retrieve it's links from .well-known/core. In order to use this method, the links must first be exposed in the .well-known/core of the pubsub broker. See Section 4.1 in this document.

The pubsub broker triggers the RD to retrieve its links by sending a POST with an empty payload to the .well-known/core of the Resource Directory. The RD server will then retrieve the links from the .well-known/core of the pubsub broker and incorporate them into the Resource Directory. See [I-D.ietf-core-resource-directory] for further details.

6. Sleep-Wake Operation

CoAP pubsub provides a way for client nodes to sleep between operations, conserving energy during idle periods. This is made possible by shifting the server role to the broker, allowing the broker to be always-on and respond to requests from other clients while a particular client is sleeping.

For example, the broker will retain the last state update received from a sleeping client, in order to supply the most recent state update to other clients in response to read and subscribe operations.

Likewise, the broker will retain the last state update received on the topic such that a sleeping client, upon waking, can perform a read operation to the broker to update its own state from the most recent system state update.

7. Simple Flow Control

Since the broker node has to potentially send a large amount of notification messages for each publish message and it may be serving a large amount of subscribers and publishers simultaneously, the broker may become overwhelmed if it receives many publish messages to popular topics in a short period of time.

If the broker is unable to serve a certain client that is sending publish messages too fast, the broker MUST respond with Response Code 4.29, "Too Many Requests". This Response Code is like HTTP 429 "Too

Many Requests" but uses the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry. The broker MAY stop creating notifications from the publish messages from this client and to this topic for the indicated time.

If a client receives the 4.29 Response Code from the broker for a publish message to a topic, it MUST NOT send new publish messages to the broker on the same topic before the time indicated in Max-Age has passed.

8. Security Considerations

CoAP pubsub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP pubsub broker and the clients SHOULD authenticate each other and enforce access control policies. A malicious client could subscribe to data it is not authorized to or mount a denial of service attack against the broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP pubsub broker introduces challenges for the use of end-to-end security between for example a client device on a sensor network and a client application running in a cloud-based server infrastructure since brokers terminate the exchange. While running separate DTLS sessions from the client device to the broker and from broker to client application protects 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].

When only end-to-end encryption is necessary and the CoAP Broker is trusted, Payload Only Protection (Mode:PAYL) could be used. The Publisher would wrap only the payload before sending it to the broker and set the option Content-Format to application/smpayl. Upon receival, the Broker can read the unencrypted CoAP header to forward it to the subscribers.

9. IANA Considerations

This document registers one attribute value in the Resource Type (rt=) registry established with [RFC6690] and appends to the definition of one CoAP Response Code in the CoRE Parameters Registry.

9.1. Resource Type value 'core.ps'

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

9.2. Response Code value '2.04'

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

9.3. Response Code value '4.29'

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

10. Acknowledgements

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

11. References

11.1. Normative References

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, D., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-07 (work in progress), March 2016.

[I-D.selander-ace-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-05 (work in progress), July 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

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

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

11.2. Informative References

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

Authors' Addresses

Michael Koster
SmartThings

Email: Michael.Koster@smartthings.com

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson

Email: jaime.jimenez@ericsson.com

core
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2015

K. Li
Alibaba Group
R. Sun
Huawei Technologies
A. Rahman
InterDigital
March 9, 2015

CBOR Equivalents of CoRE JSON Formats
draft-li-core-cbor-equivalents-00

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

Copyright Notice

Table of Contents

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:

name	value
href	resource URI
attribute name 1	attribute value 1
attribute name 2	attribute value 2
...	...
attribute name n	attribute value n

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:

name	encoded value
href	1
rel	2
anchor	3
rev	4
hreflang	5
media	6
title	7
type	8
rt	9
if	10
sz	11
ct	12

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:

o the outer collection to an array of links (Major type 4)

o 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"
```

Figure 3: Example from page 15 of [RFC6690]

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)
         2f73656e736f72732f74656d70   # "/sensors/temp"
      09                              # unsigned integer(value:9,"rt")
      6d                              # text string(13 bytes)
         74656d70657261747572652d63   # "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")
```

```
      78 23                            # text string(35 bytes)
         687474703a2f2f7777772e6578616d706c652e636f6d2f73656e736f72732f74313233
                                                # "http://www.example.com/sensors
/t123"
      03                               # unsigned integer(value:3,"anchor")
      6d                               # text string(13 bytes)
         2f73656e736f72732f74656d70    # "/sensors/temp"
      02                               # unsigned integer(value:2,"rel")
      6b                               # text string(11 bytes)
         6465736372696265646279        # "describedby"
   a3                                  # map(number of pairs of data items:3)
      01                               # unsigned integer(value:1,"href")
      62                               # text string(12 bytes)
         2f74                          # "/t"
      03                               # unsigned integer(value:3,"anchor")
      6d                               # text string(13 bytes)
         2f73656e736f72732f74656d70    # "/sensors/temp"
      02                               # unsigned integer(value:2,"rel")
      69                               # text string(9 bytes)
         616c7465726e617465            # "alternate"
```

Figure 4: Web Links Encoded in CBOR

2.4.2. Link Format in JSON to CBOR Example

This examples shows conversion from link format JSON to CBOR format.

```
[{"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                                 # 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)
         2f73656e736f72732f74656d70     # "/sensors/temp"
      09                                # unsigned integer(value:9,"rt")
      6d                                # text string(13 bytes)
         74656d70657261747572652d63     # "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")
      78 23                             # text string(35 bytes)
         687474703a2f2f7777772e6578616d706c652e636f6d2f73656e736f72732f74313233
                                                # "http://www.example.com/sensors
/t123"
      03                                # unsigned integer(value:3,"anchor")
      6d                                # text string(13 bytes)
         2f73656e736f72732f74656d70     # "/sensors/temp"
      02                                # unsigned integer(value:2,"rel")
      6b                                # text string(11 bytes)
         6465736372696265646279         # "describedby"
   a3                                   # map(number of pairs of data items:3)
      01                                # unsigned integer(value:1,"href")
      62                                # text string(12 bytes)
         2f74                           # "/t"
      03                                # unsigned integer(value:3,"anchor")
      6d                                # text string(13 bytes)
         2f73656e736f72732f74656d70     # "/sensors/temp"
      02                                # unsigned integer(value:2,"rel")
      69                                # text string(9 bytes)
         616c7465726e617465             # "alternate"
```

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:f
7fe:ed37:14ca]"
   62                                   # text(2)
      3131                              # "11"
   a2                                   # map(2)
      61                                # text(1)
         6e                             # "n"
      78 25                             # text(37)
         73656e736f72732e666c6f6f72312e776573742e626c6467362e6578616d706c652e636
f6d # "sensors.floor1.west.bldg6.example.com"
      61                                # text(1)
         61                             # "a"
      78 1b                             # text(27)
         5b666631353a3a343230303a663766653a656433373a323563625d # "[ff15::4200:f
7fe:ed37:25cb]"
   62                                   # text(2)
      3132                              # "12"
   a2                                   # map(2)
      61                                # text(1)
         6e                             # "n"
      78 29                             # text(41)
         416c6c2d446576696365732e666c6f6f72312e776573742e626c6467362e6578616d706
c652e636f6d # "All-Devices.floor1.west.bldg6.example.com"
      61                                # text(1)
         61                             # "a"
      78 20                             # text(32)
         5b666631353a3a343230303a663766653a656433373a616263645d3a34353637 # "[ff
15::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.

Thanks to Carsten Bormann for helpful comments and discussions that have shaped the document.

7. References

7.1. Normative References

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

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

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

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

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

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

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

Ruinan Sun
Huawei Technologies
Huawei Base, Bantian, Longgang District
Shenzhen, Guangdong 518129
P. R. China

Phone: +86-755-28970171
Email: sunruinan@huawei.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal, Quebec H3A 3G4
Canada

Phone: +1-514-585-0761
Email: akbar.rahman@interdigital.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 18, 2016

T. Savolainen
Nokia
K. Hartke
Universitaet Bremen TZI
B. Silverajan
Tampere University of Technology

June 16, 2016

CoAP over WebSockets
draft-savolainen-core-coap-websockets-07

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on December 18, 2016.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web protocol designed for communications between resource constrained nodes. By default, CoAP operates as a layer on top of UDP or DTLS, but there is interest in using CoAP also over other types of transports, such as TCP, TLS [I-D.ietf-core-coap-tcp-tls], or SMS [I-D.becker-core-coap-sms-gprs].

An interesting transport for CoAP could be the WebSocket Protocol [RFC6455]. The WebSocket protocol provides two-way communication between a client and a server after upgrading an HTTP/1.1 [RFC7230] connection, and may be available in an environment that does not allow transportation of CoAP over UDP. This environment can be, for example, a corporate network with Internet access only via an HTTP proxy, or a CoAP application running inside a web browser without access to connectivity means other than HTTP and WebSockets.

This document specifies how to access resources using CoAP requests and responses over the WebSocket Protocol. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either by communicating CoAP directly with a CoAP server accessible over a WebSocket Connection or via a CoAP intermediary that proxies CoAP

requests and responses between different transports, such as between WebSockets and UDP.

```
+-----------------------------------------------------------+
|                                                           |
|                         Application                       |
|                                                           |
+-----------------------------------------------------------+
|                                                           |
|                            CoAP                           |
|                   Requests and Responses                  |
|                                                           |
+ - - - - - - - - - +-------------------+-------------------+
|                  |                   |                   |
|      CoAP        |     CoAP over     |     CoAP over     |
|    Messaging     |    TCP and TLS    |    WebSockets     |
|                  |                   |                   |
+---------+--------+---------+---------+-------------------+
|         |        |         |         |                   |
|   UDP   |  DTLS  |   TCP   |   TLS   |    WebSockets     |
|         |        |         |         |                   |
+---------+--------+---------+---------+-------------------+
```

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.

```
 __________                                 ___________
|          |                               |           |
|          _|___      requests         ___|_          |
|   CoAP  /  \  \  -------------->  /  /  \  CoAP     |
|  Client \__/__/  <--------------  \__\__/ Server    |
|          |          responses            |           |
|__________|                               |___________|
        WebSocket  ============>  WebSocket
          Client     Connection     Server
```

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"
                                        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.

```
 ___________              ___________                ___________
|           |            |           |              |           |
|          _|___        ___|_        _|___        ___|_          |
|   CoAP  /  \  \ ---> /  /  \ CoAP  /  \  \ ---> /  /  \  CoAP  |
|  Client \__/__/ <--- \__\__/ Proxy \__/__/ <--- \__\__/ Server |
|           |            |           |              |           |
|___________|            |___________|              |___________|
        WebSocket ===> WebSocket      UDP            UDP
          Client        Server      Client          Server
```

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

A third possible configuration is a CoAP server running inside a web browser (Figure 5). The web browser initially connects to a WebSocket endpoint and is then reachable through the WebSocket server. When no connection exists, the CoAP server is not reachable; it therefore can be considered a Sleepy Endpoint (SEP) [I-D.dijk-core-sleepy-reqs]. Because the WebSocket server is the

only way to reach the CoAP server, the CoAP proxy should be a Reverse Proxy.

```
 ___________              ___________              ___________
|           |            |           |            |           |
|          _|___        ___|_         _|___        ___|_         |
|   CoAP  /  \  \ ---> /  /  \ CoAP  /  /  \ ---> /  \  \  CoAP   |
|  Client \__/__/ <--- \__\__/ Proxy \__\__/ <--- \__/__/ Server  |
|           |            |           |            |           |
|___________|            |___________|            |___________|
          UDP            UDP      WebSocket <=== WebSocket
        Client          Server      Server        Client
```

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

Further configurations are possible, including those where a WebSocket Connection is established through an HTTP proxy.

1.2. Terminology

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455] and [RFC7252].

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

2. CoAP over WebSockets

CoAP over WebSockets is intentionally very similar to CoAP as defined over UDP. Therefore, instead of presenting CoAP over WebSockets as a new protocol, this document specifies it as a series of deltas from [RFC7252].

2.1. Opening Handshake

Before CoAP requests and responses can be exchanged, a WebSocket Connection needs to be established as defined in Section 4 of [RFC6455]. Figure 6 shows an example.

The WebSocket client MUST include the subprotocol name "coap" in the list of protocols, which indicates support for the protocol defined in this document. Any later, incompatible versions of CoAP or CoAP over WebSockets will use a different subprotocol name.

The WebSocket client includes the hostname of the WebSocket server in the Host header field of its handshake as per [RFC6455]. The Host

header field also indicates the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server.

```
GET /.well-known/coap HTTP/1.1
Host: example.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: coap
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: coap
```

Figure 6: Example of an Opening Handshake

2.2. Message Format

Once a WebSocket Connection has been established, CoAP requests and responses can be exchanged as WebSocket messages. Since CoAP uses a binary message format, the messages are transmitted in binary data frames as specified in Sections 5 and 6 of [RFC6455].

The message format is very similar to the format specified for CoAP over UDP [RFC7252]. The differences are as follows:

- o Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. This means the "T" and "Message ID" fields in the CoAP message header can be elided.

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   R   |  TKL  |      Code     |    Token (TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: CoAP Message Format over WebSockets

The resulting message format is shown in Figure 7. The four most-significant bits of the first byte are reserved (R) and MUST be set to zero. The remaining fields and structure are the same as defined in [RFC7252].

Requests and response messages can be fragmented as specified in Section 5.4 of [RFC6455], though typically they are sent unfragmented as they tend to be small and fully buffered before transmission. The WebSocket protocol does not provide means for multiplexing; if it is not desirable for a large message to monopolize the connection, requests and responses can be transferred in a blockwise fashion as defined in [I-D.ietf-core-block].

Messages MUST NOT be Empty (Code 0.00), i.e., messages always carry either a request or a response.

2.3. Message Transmission

CoAP requests and responses are exchanged asynchronously over the WebSocket Connection, i.e., a CoAP client can send multiple requests without waiting for a response and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Concurrent requests are differentiated by their Token, which are scoped locally to the connection.

The connection is bi-directional, so requests can be sent both by the entity that established the connection and the remote host.

Retransmission and deduplication of messages is provided by the WebSocket Protocol. CoAP over WebSockets therefore does not make a distinction between Confirmable or Non-Confirmable messages, and does not provide Acknowledgement or Reset messages.

Since the WebSocket Protocol provides ordered delivery of messages, the mechanism for reordering detection when observing resources [RFC7641] is not needed. The value of the Observe Option in notifications therefore MAY be empty on transmission and MUST be ignored on reception.

2.4. Connection Health

When a client does not receive any response for some time after sending a CoAP request (or, similarly, when a client observes a resource and it does not receive any notification for some time), the connection between the WebSocket client and the WebSocket server may be lost or temporarily disrupted without the client being aware of it.

To check the health of the WebSocket Connection (and thereby of all active requests, if any), the client can send a Ping frame or an unsolicited Pong frame as specified in Section 5.5 of [RFC6455]. There is no way to retransmit a request without creating a new one. Re-registering interest in a resource is permitted, but entirely unnecessary.

2.5. Closing the Connection

The WebSocket Connection is closed as specified in Section 7 of [RFC6455].

All requests for which the CoAP client has not received a response yet are cancelled when the connection is closed. If the client observes one or more resources over the WebSocket Connection, then the CoAP server (or intermediary in the role of the CoAP server) MUST remove all entries associated with the client from the lists of observers when the connection is closed.

3. CoAP over WebSockets URIs

For the first configuration discussed in Section 1.1, this document defines two new URIs schemes that can be used for identifying CoAP resources and providing a means of locating these resources: "coap+ws" and "coap+wss".

Similar to the "coap" and "coaps" schemes, the "coap+ws" and "coap+wss" schemes organize resources hierarchically under a CoAP origin server. The key difference is that the server is potentially reachable on a WebSocket endpoint instead of a UDP endpoint.

The WebSocket endpoint is identified by a "ws" or "wss" URI that is composed of the authority part of the "coap+ws" or "coap+wss" URI, respectively, and the well-known path "/.well-known/coap" [RFC5785]. The path and query parts of a "coap+ws" or "coap+wss" URI identify a resource within the specified endpoint which can be operated on by the methods defined by the CoAP protocol.

The syntax of the "coap+ws" and "coap+wss" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty" and "query" are the same as in [RFC3986].

```
coap-ws-URI =
   "coap+ws:" "//" host [ ":" port ] path-abempty [ "?" query ]

coap-wss-URI =
   "coap+wss:" "//" host [ ":" port ] path-abempty [ "?" query ]
```

The port component is OPTIONAL; the default for "coap+ws" is port 80, while the default for "coap+wss" is port 443.

Fragment identifiers are not part of the request URI and thus MUST NOT be transmitted in a WebSocket handshake or in the URI options of a CoAP request.

3.1. Decomposing and Composing URIs

The steps for decomposing a "coap+ws" or "coap+wss" URI into CoAP options are the same as specified in Section 6.4 of [RFC7252] with the following changes:

- o The <scheme> component MUST be "coap+ws" or "coap+wss" when converted to ASCII lowercase.

- o A Uri-Host Option MUST only be included in a request when the <host> component does not equal the uri-host component in the Host header field in the WebSocket handshake.

- o A Uri-Port Option MUST only be included in a request if |port| does not equal the port component in the Host header field in the WebSocket handshake.

The steps to construct a URI from a request's options are changed accordingly.

4. Security Considerations

CoAP over WebSockets and CoAP over TLS-secured WebSockets do not introduce additional security issues beyond CoAP and DTLS-secured CoAP respectively [RFC7252].

The security considerations of [RFC6455] apply.

5. IANA Considerations

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

5.1. URI Scheme Registrations

5.1.1. "coap+ws"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+ws".

URI scheme name.

coap+ws

Status.
Permanent.

URI scheme syntax.
Defined in Section 3 of [RFCXXXX].

URI scheme semantics.
The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol.

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Interoperability considerations.
None.

Security considerations.
See Section 4 of [RFCXXXX].

Contact.
IETF Chair <chair@ietf.org>

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

References.
[RFCXXXX]

5.1.2. "coap+wss"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+wss".

URI scheme name.
coap+wss

Status.
Permanent.

URI scheme syntax.
Defined in Section 3 of [RFCXXXX].

URI scheme semantics.
The "coap+wss" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol secured with Transport Layer Security (TLS).

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Interoperability considerations.
None.

Security considerations.
See Section 4 of [RFCXXXX].

Contact.
IETF Chair <chair@ietf.org>

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

References.
[RFCXXXX]

5.2. WebSocket Subprotocol Registration

This document requests the registration of the subprotocol name "coap" in the WebSocket Subprotocol Name Registry.

Subprotocol Identifier.
coap

Subprotocol Common Name.
Constrained Application Protocol (CoAP)

Subprotocol Definition.
[RFCXXXX]

5.3. Well-Known URI Suffix Registration

This document requests the registration of the Well-Known URI suffix "coap" in the Well-Known URI Registry.

URI suffix.
coap

Change controller.
IETF

Specification document(s).
[RFCXXXX]

Related information.
None.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <http://www.rfc-editor.org/info/rfc5785>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <http://www.rfc-editor.org/info/rfc6455>.

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

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

6.2. Informative References

[I-D.becker-core-coap-sms-gprs] Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-05 (work in progress), August 2014.

[I-D.dijk-core-sleepy-reqs] Dijk, E., "Sleepy Devices using CoAP - Requirements", draft-dijk-core-sleepy-reqs-00 (work in progress), June 2013.

[I-D.ietf-core-block] Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-20 (work in progress), April 2016.

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

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <http://www.rfc-editor.org/info/rfc6454>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

Appendix A. Examples

This section gives examples for the first two configurations discussed in Section 1.1.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be

as follows. Figure 8 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap+ws://example.org/sensors/temperature?u=Cel>, for example, from a resource representation that it retrieved previously.

2. It establishes a WebSocket Connection to the endpoint URI composed of the authority "example.org" and the well-known path "/.well-known/coap", <ws://example.org/.well-known/coap>.

3. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.

4. It waits for the server to return a response.

5. The CoAP client uses the connection for further requests, or the connection is closed.

```
      CoAP        CoAP
     Client      Server
   (WebSocket  (WebSocket
     Client)     Server)

        |           |
        |           |
        +=========>|  GET /.well-known/coap HTTP/1.1
        |           |  Host: example.org
        |           |  Upgrade: websocket
        |           |  Connection: Upgrade
        |           |  Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
        |           |  Sec-WebSocket-Protocol: coap
        |           |  Sec-WebSocket-Version: 13
        |           |
        <=========+  HTTP/1.1 101 Switching Protocols
        |           |  Upgrade: websocket
        |           |  Connection: Upgrade
        |           |  Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
        |           |  Sec-WebSocket-Protocol: coap
        |           |
        |           |
        +--------->|  Binary frame (opcode=%x2, FIN=1, MASK=1)
        |           |    +-------------------------+
        |           |    | GET                     |
        |           |    | Token: 0x53             |
        |           |    | Uri-Path: "sensors"     |
        |           |    | Uri-Path: "temperature" |
        |           |    | Uri-Query: "u=Cel"      |
        |           |    +-------------------------+
        |           |
        <---------+  Binary frame (opcode=%x2, FIN=1, MASK=0)
        |           |    +-------------------------+
        |           |    | 2.05 Content            |
        |           |    | Token: 0x53             |
        |           |    | Payload: "22.3 Cel"     |
        |           |    +-------------------------+
        :           :
        :           :
        |           |
        +--------->|  Close frame (opcode=%x8, FIN=1, MASK=1)
        |           |
        <---------+  Close frame (opcode=%x8, FIN=1, MASK=0)
        |           |
```

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.

```
   CoAP        CoAP       CoAP
  Client      Proxy      Server
 (WebSocket  (WebSocket    (UDP
  Client)     Server)   Endpoint)

      |           |           |
      +--------->|           |  Binary frame (opcode=%x2, FIN=1, MASK=1)
      |           |           |    +-------------------------------------+
      |           |           |    | GET                                 |
      |           |           |    | Token: 0x7d                         |
      |           |           |    | Proxy-Uri: "coap://[2001:DB8::1]/"  |
      |           |           |    +-------------------------------------+
      |           |           |
      |           +--------->|  CoAP message (Ver=1, T=Con, MID=0x8f54)
      |           |           |    +-------------------------------------+
      |           |           |    | GET                                 |
      |           |           |    | Token: 0x0a15                       |
      |           |           |    +-------------------------------------+
      |           |           |
      |           <---------+  CoAP message (Ver=1, T=Ack, MID=0x8f54)
      |           |           |    +-------------------------------------+
      |           |           |    | 2.05 Content                        |
      |           |           |    | Token: 0x0a15                       |
      |           |           |    | Payload: "ready"                    |
      |           |           |    +-------------------------------------+
      |           |           |
      <---------+           |  Binary frame (opcode=%x2, FIN=1, MASK=0)
      |           |           |    +-------------------------------------+
      |           |           |    | 2.05 Content                        |
      |           |           |    | Token: 0x7d                         |
      |           |           |    | Payload: "ready"                    |
      |           |           |    +-------------------------------------+
      |           |           |
```

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

Acknowledgements

Thanks to Nadir Javed for helpful comments and discussions that have shaped the document.

Authors' Addresses

Teemu Savolainen
Nokia
Hermiankatu 12 D
Tampere FI-33720
Finland

Email: teemu.savolainen@nokia.com

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

Phone: +49-421-218-63905
Email: hartke@tzi.org

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

ACE Working Group
Internet-Draft
Intended Status: Standards Track
Expires: September 10, 2015

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

March 9, 2015

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).

Table of Contents

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:

- o 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.)

- o 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].

- o 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.

- o 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.

No.	C	U	N	R	Name	Format	Length *)
TBD	x		x		Sig	opaque	12-TBD

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

Table 1: The Sig Option

*) Length is essentially Length(SSM Header) + Length(SSM Tag). The minimum length is estimated in Appendix C. The maximum length depends on actual message format selected and is TBD.

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:

- o the 8-bit CoAP header field Code;
- o 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
- o 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.

No.	C	U	N	R	Name	Format	Length *)
TBD	x		x		Enc	opaque	0 or 12-TBD

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

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 describe 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:

- o 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

- o 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:

- o the 8-bit CoAP header field Code;

- o 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:

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

- o 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).

- o 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

Mode:APPL object is indicated by setting the option Content-Format to application/sm. A CoAP client may request a response containing such a payload wrapping by setting the option Accept to application/sm. (See Section 8.)

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:

Number	Name	Reference
TBD	Sig	[[this document]]
TBD	Enc	[[this document]]

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]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-41 (work in progress), January 2015.

[I-D.ietf-jose-json-web-encryption]
Jones, M., and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-40 (work in progress), January 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

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

10.2 Informative References

[I-D.seitz-ace-problem-description]
Seitz, L., and G. Selander, "Problem Description for Authorization in Constrained Environments", draft-seitz-ace-problem-description-02 (work in progress), May 2015.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-02 (work in progress), February 2015.

[I-D.bormann-jose-cose]
Bormann, C., "Constrained Object Signing and Encryption

(COSE)", draft-bormann-jose-cose-00 (work in progress, October 2014

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

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

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

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

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:

- o 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.

- o 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:

  - o All Uri-* options removed

  - o 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:

- o Max-Age, Observe: This information is intended to be read by a proxy.
- o Enc, Sig: These are the security-providing options.
- o Uri-Host, Uri-Port: This information can be inferred from destination IP address and port.
- o 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.

No.	C	U	N	R	Name	Format	Length	E	IP
1	x			x	If-Match	opaque	0-8	x	x
3	x	x	-		Uri-Host	string	1-255		a
4				x	ETag	opaque	1-8	x	x
5	x				If-None-Match	empty	0	x	x
6		x	-		Observe	uint	0-3		

7	x	x	-		Uri-Port	uint	0-2		a
8				x	Location-Path	string	0-255	x	x
11	x	x	-	x	Uri-Path	string	0-255	x	b
12					Content-Format	uint	0-2	x	x
14		x	-		Max-Age	uint	0-4		
15	x	x	-	x	Uri-Query	string	0-255	x	b
17	x				Accept	uint	0-2	x	x
20				x	Location-Query	string	0-255	x	x
35	x	x	-		Proxy-Uri	string	1-1034		x
39	x	x	-		Proxy-Scheme	string	1-255		
60			x		Size1	uint	0-4	x	x

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

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

- o the Parameter "kid" defined in Section 4.1.4 of [I-D.ietf-jose-json-web-signature];
- o the new Parameter "seq" defined in B.3; and
- o 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

- o the Parameter "kid" defined in Section 4.1.4 of [I-D.ietf-jose-json-web-signature]; and

- o 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| M |    ALG    |   KL    |  SL |             KID               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                             SEQ                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                             Body                              ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                             Tag                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Compact Secure Message format

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):

- o 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.

- o Algorithm (ALG). This parameter consists of an encoding of the ciphersuite used in the Secure Message. The encoding is TBD.

- o 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.

- o 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.

- o 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).

- o 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).

Name	Parameter	Length
M	Mode	2 bits
ALG	Algorithm	6 bits
KL	Key Identifier Length	5 bits
SL	Sequence Number Length	3 bits
KID	Key Identifier	KL + 1: 1-32 bytes
SEQ	Sequence Number	SL + 1: 1-8 bytes

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.

Scheme	Header	MAC	Total Overhead
JWS	90 B	43 B	135 bytes
COSE	35 B	32 B	70 bytes
CSM	12 B	16 B	28 bytes

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.

| Scheme | Header | IV | Tag | Total Overhead |

JWE	86 B	16 B	22 B	127 bytes
COSE	40 B	12 B	16 B	70 bytes
CSM	12 B	0 B	8 B	20 bytes

Table 7: Comparison of JWE, COSE, and CSM

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
    |      |      |
    |      |      |
    |      |      |
    +----->|      |      Code: 0.03 (PUT)
    | PUT  |      |     Token: 0x8c
    |      |      | Uri-Path: lock
    |      |      |      Sig: SSM {"mod"="1","seq":"00000142",
    |      |      |           "kid":"a1534e3c5fdc09bd", ...}
    |      |      |  Payload: 1
    |      |      |
    |      +----->|      Code: 0.03 (PUT)
    |      | PUT  |    Token: 0x7b
    |      |      | Uri-Path: lock
    |      |      |      Sig: SSM {"mod"="1","seq":"00000142",
```

```
|       |       |            "kid":"a1534e3c5fdc09bd", ...}
|       |       |   Payload: 1
|       |       |
|       |       |
|       |<-----+      Code: 2.04 (Changed)
|       | 2.04 |     Token: 0x7b
|       |       |       Sig: SSM {"mod"="1","seq":"000000a6",
|       |       |            "kid":"5fdc09bda1534e3c", ...}
|       |       |
|<-----+        |      Code: 2.04 (Changed)
| 2.04 |        |     Token: 0x8c
|       |       |       Sig: SSM {"mod"="1","seq":"000000a6",
|       |       |            "kid":"5fdc09bda1534e3c", ...}
|       |       |
```

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
    |      |      |
    |      |      |
```

```
       |       |       |
       |       |       |
       +------>|       |      Code: 0.01 (GET)
       | GET   |       |     Token: 0x83
       |       |       |       Enc: SEM {"mod"="1","seq":"000015b7",
       |       |       |            "kid":"34e3c5fdca1509bd",
       |       |       |            ["glucose" ... ], ...}
       |       |       |
       |       |       |
       |       +------>|      Code: 0.01 (GET)
       |       | GET   |     Token: 0xbe
       |       |       |       Enc: SEM {"mod"="1","seq":"000015b7",
       |       |       |            "kid":"34e3c5fdca1509bd",
       |       |       |            ["glucose" ... ], ...}
       |       |       |
       |       |       |
       |       |       |
       |       |       |
       |       |<------+      Code: 2.05 (Content)
       |       | 2.05  |     Token: 0xbe
       |       |       |       Enc:
       |       |       |   Payload: SEM {"mod"="1","seq":"000015b7",
       |       |       |            "kid":"c09bda155fd34e3c",
       |       |       |            [... 220], ...}
       |       |       |
       |       |       |
       <------+        |      Code: 2.05 (Content)
       | 2.05  |       |     Token: 0x83
       |       |       |       Enc:
       |       |       |   Payload: SEM {"mod"="1","seq":"000015b7",
       |       |       |            "kid":"c09bda155fd34e3c",
       |       |       |            [... 220], ...}
       |       |       |
       |       |       |
```

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)
     | GET  |         Token: 0x83
     |      |      Proxy-Uri: example.com/temp
     |      |      |
     |      |      |
     |      +----->|      Code: 0.01 (GET)
     |      | GET         Token: 0xbe
     |      |      Uri-Host: example.com
     |      |      Uri-Path: temp
     |      |      |
     |      |      |
     |      |<-----+      Code: 2.05 (Content)
     |      | 2.05        Token: 0xbe
     |      |      Payload: SEM {"mod"="0","seq":"000015b7",
     |      |               "kid":"c09bda155fd34e3c",
     |      |               ["471 F"], ...}
     |      |      |
     |<-----+      |      Code: 2.05 (Content)
     | 2.05 |         Token: 0x83
     |      |      Payload: SEM {"mod"="0","seq":"000015b7",
     |      |               "kid":"c09bda155fd34e3c",
     |      |               ["471 F"], ...}
Client2     |      |
            |      |
     |      |      |
     |      |      |
     +----->|      |      Code: 0.01 (GET)
```

```
       | GET  |       |     Token: 0xa1
       |      |       | Proxy-Uri: example.com/temp
       |      |       |
       |<-----+       |      Code: 2.05 (Content)
       | 2.05 |       |     Token: 0xa1
       |      |       |   Payload: SEM {"mod"="0","seq":"000015b7",
       |      |       |            "kid":"c09bda155fd34e3c",
       |      |       |            ["471 F"], ...}
```

Figure 10: Proxy caching protected with SEM (Mode:APPL)

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-    PubSub- Publisher
scriber Broker

       |      |
       +----->|       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"], ...}
       |      |
       |      |
       |      |
       |      |<-----+      Code: 0.03 (PUT)
       |      | PUT  |      Token: 0x1f
       |      |      |   Uri-Path: ps
       |      |      |   Uri-Path: birch-pollen
       |      |      |    Payload: SSM {"mod"="0","seq":"000015b8",
```

```
|       |       |            "kid":"c09bda155fd34e3c",
|       |       |            ["300"], ...}
|       |       |
|       +------>|      Code: 2.04 (Changed)
|        2.04   |     Token: 0x1f
|       |       |
|       |       |
|<------+       |      Code: 2.05 (Content)
| 2.05  |       |     Token: 0x72
|       |       |  Observe: 2
|       |       |  Payload: SSM {"mod"="0","seq":"000015b8",
|       |       |            "kid":"c09bda155fd34e3c",
|       |       |            ["300"], ...}
```

Figure 11: Publish-subscribe protected with SSM (Mode:APPL)

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
```

```
       |       |       |
       |       |       |
       |       |       |       Client access Access Token:
       +- - ->|       |     SEM {"mod"="0","seq":"00000142",
       |       |       |          "kid":"c09bda1534e3c5fdc09bd", ...}
       |       |       |
       |       |       |
       |       +------>|      Code: 0.03 (PUT)
       |       | PUT   |     Token: 0xac
       |       |       |  Uri-Path: authorization
       |       |       |   Payload: SEM {"mod"="0","seq":"00000142",
       |       |       |            "kid":"c09bda1534e3c5fdc09bd", ...}

   a)
       |       |       |
       |       |<------+      Code: 2.04 (Changed)
       |       | 2.04  |     Token: 0xac
       |       |       |

   b)
       |       |       |
       |       |<------+      Code: 4.01 (Unauthorized)
       |       | 4.01  |     Token: 0xac
       |       |       |
```

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
Scheelevagen 17
22370 Lund
SWEDEN

EMail: ludwig@sics.se

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

B. Silverajan
Tampere University of Technology
T. Savolainen
Nokia Technologies

March 5, 2018

CoAP Communication with Alternative Transports
draft-silverajan-core-coap-alternative-transports-11

Abstract

The aim of this document is to provide a way forward to best decide upon how alternative transport information can be expressed in a CoAP URI. This draft examines the requirements for a new URI format for representing CoAP resources over alternative transports. Various potential URI formats are presented. Benefits and drawbacks of embedding alternative transport information in various ways within the URI components are also discussed. From all listed formats, the document finds scheme-based model to be the most technically feasible.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a lightweight, binary application layer protocol designed for constrained environments. Owing to its operating environment, CoAP uses UDP and DTLS as its underlying transports between communicating endpoints. However, with an increase in deployment experiences as well as its popularity, compelling reasons exist for extending CoAP messaging to work over alternative transports. These allow CoAP to better address firewall and NAT traversal issues, to operate in Web browser-based and HTML5 applications as well as for energy-constrained M2M communication in cellular networks. At the time of writing, these transports are:

- TCP, TLS and Websockets [RFC8323]

- o SMS for cellular networks[I-D.becker-core-coap-sms-gprs]

- o SLIP for serial interfaces[I-D.bormann-t2trg-slipmux]

CoAP uses a REST-based model similar to HTTP, where URIs are used to identify resources at servers. An important factor of allowing CoAP communication over alternative transports, is to express not only the resource identifier, but also the alternative transport information in the URI.

CoAP URIs contain information, such as the endpoint address as well as the location of the resource hosted at the endpoint. CoAP URIs beginning with "coap://" are using UDP, while those beginning with "coaps://" are using DTLS.

```
   coap :// server.example.org /sensors/temperature
   \___/    \______  ________/  \______  __________/
     |             \/                  \/
 URI scheme   URI authority         URI path
```

Figure 1: A CoAP URI

Figure 1 shows the structure of a simple example CoAP URI, in which the various URI components can beinterpreted as follows:

- o The URI scheme component (e.g. "coap") contains an application-level identifier which typically identifies the protocol being used as well as its transport and network level protocol configurations. Such configurations are defined by convention or standardisation of the protocol using the scheme.

- o The URI authority component ("server.example.com") contains the endpoint identification, which is typically a fully qualified domain name or a network-level host address.

- o The URI path component ("/sensors/temperature") contains a parameterised resource identifier providing the location and identity of the resource at the endpoint.

In addition to these URI components, Figure 2 shows how specific queries on resource representations are provided by CoAP clients to servers, by specifying one or more URI query components in the URI.

```
coap :// server.example.org /sensors/temperature ?u=cel
                                                  \___/
                                                    |
                                                 URI query
```

Figure 2: A CoAP URI with query

This document focuses on how CoAP URIs can be extended to contain information about alternative transports. For deriving the new URI format, the main design considerations are presented in the next section. Following that, various potential URIs are presented. These URIs provide examples of how transport identifiers can be situated in the URI scheme, authority, path or query components. The proposed URIs are analysed to select feasible formats while disqualifying those not meeting the design criteria.

2. Conformance and Design Considerations

In order to understand which URI formats are best suited for expressing transport information, certain considerations firstly need to be taken into account. Doing so eliminates URI formats that do not meet or conform to the stated requirements. The main criteria are:

1. Conformance to the generic syntax for a URI described in [RFC3986]. A URI format needs to be described in which each URI component clearly meets the syntax and percent-encoding rules described.

2. Alignment with best practices for URI design, as described in [RFC7320]. This is particularly important when it pertains to establishing or standardising the structure and usage of URIs with respect to the various URI components.

3. Request messages sent to a CoAP endpoint using a CoAP Transport URI may be responded to with a relative URI reference. [RFC3986] provides an algorithm to establish how relative references can be resolved against a base URI to obtain a target URI. Given this algorithm, a URI format needs to be described in which relative reference resolution does not result in a target URI that loses its transport-specific information

4. The URI can be supplied as a Proxy-Uri option by a CoAP end-point to a CoAP forward proxy. This allows communication with a CoAP end-point residing in a network using a different transport. Section 6.4 of [RFC7252] provides an algorithm for parsing a received URI to obtain the request's options. Conformance to

[RFC3986] is also necessary in order for the parsing algorithm to be successful.

In addition to the above mentioned requirements, where possible, the following considerations need to be borne in mind:

1. The URI format is able to represent a resource and the transport information for use in constrained environments, without requiring the presence of a naming infrastructure, such as DNS or a directory/lookup service.

2. Alternative transport information can be easily retrieved by computationally constrained nodes. In other words, the URI format does not result in unneccessarily complex code or logic in such nodes to parse and extract the transport to be used, nor the endpoint address.

3. URIs are designed to uniquely identify resources. When a single resource is represented with multiple URIs, URI aliasing [WWWArchv1] occurs. Avoiding URI aliasing is considered good practice.

4. CoAP URIs do not support fragment identifiers.

3. Situating Transport Information in CoAP URIs

The following subsections aim to describe potential URI formats in which the alternative transport information is placed in various URI components.

3.1. Using the URI scheme component

Expressing the transport information in the URI scheme component can be achieved by using new schemes. These can conform to an agreed-upon convention such as "coap+alternative_transport_name" for each new alternative transport and/or "coaps+alternative_transport_name" for its secure counterpart.

Examples of such URIs are:

- coap+tcp://server.example.org/sensors/temperature for using CoAP over TCP

- coap+sms://0015105550101/sensors/temperature for using CoAP over SMS with the endpoint identifier being a telephone subscriber number

- o coaps+tcp://server.example.org/sensors/temperature for using CoAP over TLS

3.1.1. Analysis

Expressing transport information in the URI scheme delivers a URI which is human-readable and computationally as easy to parse as standard CoAP URIs, to extract transport identification information. The URI syntax conforms to [RFC3986], and relative URI resolution does not result in the loss of transport identification information. However, each new alternative transport requires minting new schemes, and IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [RFC7595]. Additionally, should a CoAP server wish to expose its resources over multiple transports (such as both UDP and TCP) , URI aliasing can occur if the URI scheme components of these multiple URIs differ in describing the same resource.

3.2. Using the URI authority component

Expressing the transport information within the authority component can result in two possible URI formats.

The first approach is to structure the URI authority's host sub-component with a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint_identifier".

Examples of resulting URIs are:

- o coap://tcp-server.example.org/sensors/temperature for using CoAP over TCP

- o coap://sms-0015105550101/sensors/temperature for using CoAP over SMS

The second approach is to hint at the alternative transport information, by explicitly specifying using the URI authority's port sub-component, thereby differentiating them from standard CoAP URIs.

Examples of resulting URIs are:

- o coap://server.example.org:5684/sensors/temperature for using CoAP over TLS

- o coap://server.example.org:80/sensors/temperature for using CoAP over WebSockets

3.2.1. Analysis

Embedding the transport information in the host would violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. Consequently, the host in a URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

Embedding the transport information in the port, on the other hand, would not violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. It would result in a CoAP URI that is less human-readable, but URI aliasing is minimised.

On the other hand, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form "//server2.example.org/path/to/another/resource", relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

3.3. Using the URI path component

Should the URI path component be used, then special characters or keywords need to be supplied in the path to make the transport explicit. Here, many proposals can exist. In general however, this will result in a URI format such as:

- coap://server.example.org/sensors/temperature;tcp for using CoAP over TCP, by appending the transport information at the end of the URI.

3.3.1. Analysis

Embedding the transport information in the URI path directly results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form "../../path/to/another/resource", relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI path component cannot be used as a basis for a new CoAP URI for alternative transports.

3.4. Using the URI query component

The alternative transport information, should URI query components be used, would result in a URI format such as:

- o coap://server.example.org/sensors/temperature?alternative-transport=wss for using CoAP over secure WebSockets.

3.4.1. Analysis

Embedding the transport information in a URI query also results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form "../../path/to/another/resource", relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI query component cannot be used as a basis for a new CoAP URI for alternative transports.

4. Discussion

Based on the analysis of the various options for embedding alternative transport information in a CoAP URI, the most technically feasible option is to use the URI scheme component, as described in Section 3.1. To date, this has also been the WG consensus.

A discussion with IESG members during review of [RFC8323] revealed however, that using the URI scheme to express transport information is not desirable, to avoid the proliferation of new URI schemes for the same application-layer protocol. A strategy was instead proposed to preserve the existing CoAP URI and reuse it for alternative transports, by employing a combination of UDP Confirmable messages and timeouts to determine the eventual correct transport to use between a client and server [IESG-feedback]. The undertaken strategy would have obvious implications regarding interoperability, application and protocol logic, resource usage, for both new CoAP and existing CoAP implementations and deployments. Although URI aliasing can theoretically be avoided with this approach, at the time of writing, its technical feasibility over using the simpler strategy of using URI schemes, has yet to be validated. An obvious drawback is therefore that implementers and other SDOs may choose to provisionally or permanently register new URI schemes with IANA, for CoAP over alternative transports anyway, as was done by the Open Connectivity Foundation (OCF) [CoAP-TCP-TLS-registration].

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

New security risks are not envisaged to arise from the guidelines given in this document, for describing a new URI format containing transport identification within the URI scheme component. However, when specific alternative transports are selected for implementing support for carrying CoAP messages, risk factors or vulnerabilities can be present. Examples include privacy trade-offs when MAC addresses or phone numbers are supplied as URI authority components, or if specific URI path components employed for security-specific interpretations are accidentally encountered as false positives. While this document does not make it mandatory to introduce a security mode with each transport, it recommends ascribing meaning to the use of "coap+" and "coaps+" prefixes in the scheme component, with the "coaps+" prefix used for secure transports for CoAP messages.

7. Acknowledgements

Email discussions, comments and ideas from Thomas Fossati, Akbar Rahman, Klaus Hartke, Martin Thomson, Mark Nottingham, Dave Thaler, Graham Klyne, Carsten Bormann and Markus Becker greatly helped previous versions of this draft.

8. References

8.1. Normative References

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

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

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

8.2. Informative References

[CoAP-TCP-TLS-registration] , <https://www.iana>.

[I-D.becker-core-coap-sms-gprs]
Kuladinithi, K., Becker, M., Li, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-06 (work in progress), February 2017.

[I-D.bormann-t2trg-slipmux]
Bormann, C. and T. Kaupat, "Slipmux: Using an UART interface for diagnostics, configuration, and packet transfer", draft-bormann-t2trg-slipmux-02 (work in progress), January 2018.

[IESG-feedback]
, <https://www.ietf.org/mail-archive/web/core/current/msg08768>.

[RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates and Service: Schemes", RFC 2609, DOI 10.17487/RFC2609, June 1999, <https://www.rfc-editor.org/info/rfc2609>.

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

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

[WWWArchv1]
, <http://www.w3>.

Appendix A. Expressing transport in the URI in other ways

Other means of indicating the transport as a distinguishable component within the CoAP URI are possible, but have been deemed unsuitable by not meeting the design considerations listed, or are incompatible with existing practices outlined in [RFC7252]. They are however, retained in this section for historical documentation and completeness.

A.1. Transport information as part of the URI authority

A single URI scheme, "coap-at" can be introduced, as part of an absolute URI which expresses the transport information within the authority component. One approach is to structure the component with

a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint_identifier".

Examples of resulting URIs are:

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

- o coap-at://sms-0015105550101/sensors/temperature

An implementation note here is that some generic URI parsers will fail when encountering a URI such as "coap-at://tcp-[2001:db8::1]/sensors/temperature". Consequently, an equivalent, but parseable URI from the ip6.arpa domain needs to be formulated instead. For [2001:db8::1] using TCP, this would result in the following URL:

coap-at://tcp-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa:5683/sensors/temperature

Usage of an IPv4-mapped IPv6 address such as [::ffff.192.100.0.1] can similarly be expressed with a URI from the ip6.arpa domain.

This URI format allows the usage of a single scheme to represent multiple types of transport end-points. Consequently, it requires consistency in ensuring how various transport-specific endpoints are identified, as a single URI format is used. Attention must be paid towards the syntax rules and encoding for the URI host component. Additionally, against a base URI of the form "coap-at://tcp-server.example.com/sensors/temperature", resolving a relative reference, such as "//example.net/sensors/temperature" would result in the target URI "coap-at://example.net/sensors/temperature", in which transport information is lost.

A.1.1. Usage of DNS records

DNS names can be used instead of IPv6 address literals to mitigate lengthy URLs referring to the ip6.arpa domain, if usage of DNS is possible.

DNS SRV records can also be employed to formulate a URL such as:

coap-at://srv-_coap._tcp.example.com/sensors/temperature

in which the "srv" prefix is used to indicate that a DNS SRV lookup should be used for _coap._tcp.example.com, where usage of CoAP over TCP is specified for example.com, and is eventually resolved to a numerical IPv4 or IPv6 address.

A.2. Making CoAP Resources Available over Multiple Transports

The CoAP URI used thus far is as follows:

```
URI         = scheme ":" hier-part [ "?" query ]
hier-part   = "//" authority path-abempty
```

A new URI format could be introduced, that does not possess an "authority" component, and instead defining "hier-part" to instead use another component, "path-rootless", as specified by RFC3986 [RFC3986]. The partial ABNF format of this URI would then be:

```
URI         = scheme ":" hier-part [ "?" query ]
hier-part   = path-rootless
path-rootless = segment-nz *( "/" segment )
```

The full syntax of "path-rootless" is described in [RFC3986]. A generic URI defined this way would conform to the syntax of [RFC3986], while the path component can be treated as an opaque string to indicate transport types, endpoints as well as paths to CoAP resources. A single scheme can similarly be used.

A constrained node that is capable of communicating over several types of transports (such as UDP, TCP and SMS) would be able to convey a single CoAP resource over multiple transports. This is also beneficial for nodes performing caching and proxying from one type of transport to another.

Requesting and retrieving the same CoAP resource representation over multiple transports could be rendered possible by prefixing the transport type and endpoint identifier information to the CoAP URI. This would result in the following example representation:

```
coap-at:tcp://example.com?coap://example.com/sensors/temperature
        \_______  ______/ \_________________  ___________________/
                \/                          \/
         Transport-specific            CoAP Resource
             Prefix
```

Figure 3: Prefixing a CoAP URI with TCP transport

Such a representation would result in the URI being decomposed into its constituent components, with the CoAP resource residing within the query component as follows:

Scheme: coap-at

Path: tcp://example.com

Query: coap://example.com/sensors/temperature

The same CoAP resource, if requested over a WebSocket transport, would result the following URI:

```
coap-at:ws://example.com/endpoint?coap://example.com/sensors/temperature
        \___________ __________/ \_________________ ____________________/
                    \/                            \/
              Transport-specific            CoAP Resource
                    Prefix
```

Figure 4: Prefixing a CoAP URI with WebSocket transport

While the transport prefix changes, the CoAP resource representation remains the same in the query component:

Scheme: coap-at

Path: ws://example.com/endpoint

Query: coap://example.com/sensors/temperature

The URI format described here overcomes URI aliasing [WWWArchv1] when multiple transports are used, by ensuring each CoAP resource representation remains the same, but is prefixed with different transports. However, against a base URI of this format, resolving relative references of the form "//example.net/sensors/temperature" and "/sensor2/temperature" would again result in target URIs which lose transport-specific information.

Implementation note: While square brackets are disallowed within the path component, the '[' and ']' characters needed to enclose a literal IPv6 address can be percent-encoded into their respective equivalents. The ':' character does not need to be percent-encoded. This results in a significantly simpler URI string compared to section 2.2, particularly for compressed IPv6 addresses. Additionally, the URI format can be used to specify other similar address families and formats, such as Bluetooth addresses.

A.3. Transport as part of a 'service:' URL scheme

The "service:" URL scheme name was introduced in [RFC2609] and forms the basis of service description used primarily by the Service Location Protocol. An abstract service type URI would have the form "service:<abstract-type>:<concrete-type>"

where <abstract-type> refers to a service type name that can be associated with a variety of protocols, while the <concrete-type>

then providing the specific details of the protocol used, authority and other URI components.

Adopting the "service:" URL scheme to describe CoAP usage over alternative transports would be rather trivial. To use a previous example, a CoAP service to discover a Resource Directory and its base RD resource using TCP would take the form

service:coap:tcp://host.example.com/.well-known/core?rt=core-rd

The syntax of the "service:" URL scheme differs from the generic URI syntax and therefore such a representation should be treated as an opaque URI as Section 2.1 of [RFC2609] recommends.

Authors' Addresses

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

Teemu Savolainen
Nokia Technologies
Hatanpaeaen valtatie 30
FI-33100 Tampere
Finland

Email: teemu.savolainen@nokia.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

B. Silverajan
TUT
M. Ocak
Ericsson

July 2, 2018

CoAP Protocol Negotiation
draft-silverajan-core-coap-protocol-negotiation-09

Abstract

CoAP has been standardised as an application-level REST-based protocol. When multiple transport protocols exist for exchanging CoAP resource representations, this document introduces a way forward for CoAP endpoints as well as intermediaries to agree upon alternate transport and protocol configurations as well as URIs for CoAP messaging. Several mechanisms are proposed: Extending the CoRE Resource Directory with new parameter types, introducing a new CoAP Option with which clients can interact directly with servers without needing the Resource Directory, and finally a new CoRE Link Attribute allowing exposing alternate locations on a per-resource basis.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] allows clients, origin servers and proxies, to exchange and manipulate resource representations using REST-based methods over UDP or DTLS. CoAP messaging however can use other alternative underlying transports [I-D.silverajan-core-coap-alternative-transports].

When CoAP-based endpoints and proxies possess the ability to perform CoAP messaging over multiple transports, significant benefits can be obtained if communicating client endpoints can discover that multiple transport bindings may exist on an origin server over which CoAP resources can be retrieved. This allows a client to understand and possibly substitute a different transport protocol configuration for the same CoAP resources on the origin server, based on the preferences of the communicating peers. Inevitably, if two CoAP endpoints reside in distinctly separate networks with orthogonal transports, a CoAP proxy node is needed between the two networks so that CoAP Requests and Responses can be exchanged properly.

A URI in CoAP, however, serves two purposes simultaneously. It firstly functions as a locator, by specifying the network location of the endpoint hosting the resource, and the underlying transport used by CoAP for accessing the resource representation. It secondly identifies the name of the specific resource found at that endpoint together with its namespace, or resource path. A single CoAP URI cannot be used to express the identity of the resource independently of alternate underlying transports or protocol configuration. Multiple URIs can result for a single CoAP resource representations if:

- o the authority components of the URI differ, owing to the same physical host exposing several network endpoints. For example, "coap://example.org/sensors/temperature" and "coap://example.net/sensors/temperature"

- o the scheme components of the URI differ, owing to the origin server exposing several underlying transport alternatives. For example, "coap://example.org/sensors/temperature" and "coap+tcp://example.org/sensors/temperature"

Without a priori knowledge, clients would be unable to ascertain if two or more URIs provided by an origin server are associated to the same representation or not. Consequently, a communication mechanism needs to be conceived to allow an origin server to properly capture the relationship between these alternate representations or locations and then subsequently supply this information to clients. This also goes some way in limiting URI aliasing [WWWArchv1].

In order to support CoAP clients, proxies and servers wishing to use CoAP over multiple transports, this draft proposes the following:

- o An ability for servers to register supported CoAP transports to a CoRE Resource Directory [I-D.ietf-core-resource-directory] with optional registration lifetime values

- o A means for CoAP clients to interact with a CoRE resource directory interface for requesting and discovering alternative transports and locations of CoAP resources

- o New Resource Directory parameter types enabling the above-mentioned features.

- o A new CoAP Option called Alternative-Transport that can be used by CoAP clients to discover and retrieve the types of alternative transports available at the origin server, as well as the links describing the transport-specific endpoint address at which CoAP resources are exposed from.

- o A new CoRE Link attribute for exposing transports and endpoint locations on an origin server on a per-resource basis.

2. Aim

The following simple scenarios aim to better portray how CoAP protocol negotiation benefits communicating nodes

2.1. Overcoming Middlebox Issues

Discovering which transports are available is important for a client to determine the optimal alternative to perform CoAP messaging according to its needs, particularly when separated from a CoAP server via a NAT. It is well-known that some firewalls as well as many NATs, particularly home gateways, hinder the proper operation of UDP traffic. NAT bindings for UDP-based traffic do not have as long timeouts as TCP-based traffic.

```
+-------------+-----+      +---+      +-------------------+
|             |     |--1-->|   |--1-->|     |             |
|             | UDP |      | N |      | UDP |             |
|             |     |<--2--|   |<--2--|     |             |
| CoAP Client +-----+      | A |      +-----+ CoAP Server |
|             |     |--3-->|   |--3-->|     |             |
|             | TCP |      | T |      | TCP |             |
|             |     |<--4--|   |<--4--|     |             |
+-------------+-----+      +---+      +-----+-------------+
```

Figure 1: CoAP Client initially accesses CoAP Server over UDP and then switching to TCP

Figure 1 depicts such a scenario, where a CoAP client residing behind a NAT uses UDP initially for accessing a CoAP Server, and engages in discovering alternative transports offered by the server. The client subsequently decides to use TCP for CoAP messaging instead of UDP to set up an Observe relationship for a resource at the CoAP Server, in order to avoid incoming packets containing resource updates being discarded by the NAT.

2.2. Better resource caching and serving in proxies

Figure 2 outlines a more complex example of intermediate nodes such as CoAP-based proxies to intelligently cache and respond to CoAP or HTTP clients with the same resource representation requested over alternative transports or server endpoints. As with the earlier example, the CoAP Server registers its transports to a Resource Directory (This is assumed to be performed beforehand and not depicted in the figure, for brevity)

In this example, a CoAP over WebSockets client successfully obtains a response from a CoAP forward proxy to retrieve a resource representation from an origin server using UDP, by supplying the CoAP server's endpoint address and resource in a Proxy-URI option. Arrow 1 represents a GET request to "coap+ws://proxy.example.com" which subsequently retrieves the resource from the CoAP server using the URI "coap://example.org/sensors/temperature", shown as arrow 2.

```
+---------+
| CoAP+WS |     +--------+-------+---+     +-----+---------+
| Client  |<-1->|  Web   |       |   |<-2->|     |         |
+---------+     | Socket | CoAP  | U |     | UDP |   CoAP  |
+---------+     +--------+ Proxy | D |     +-----+  Server |
|  HTTP   |<-3->|  HTTP  |       | P |     | TCP |         |
| Client  |<-4->|        |       |   |     |     |         |
+---------+     +--------+-------+---+     +-----+---------+
```

Figure 2: Proxying and returning a resource's alternate cached representations to multiple clients

Subsequently, assume an HTTP client requests the same resource, but instead specifies a CoAP over TCP alternative URI instead. Arrow 3 represents this event, where the HTTP client performs a GET request to "http://proxy.example.com/coap+tcp://example.org/sensors/temperature". When the proxy receives the request, instead of immediately retrieving the temperature resource again over TCP, it

first verifies either from the Resource Directory or directly from the server, whether the cached resource retrieved over UDP is a valid equivalent representation of the resource requested by the HTTP client over TCP. Upon confirmation, the proxy is able to supply the same cached representation to the HTTP client as well (arrow 4).

2.3. Interaction with Energy-constrained Servers

Figure 3 illustrates discovery and communication between a CoAP client and an energy-constrained CoAP Server. Such a server aims at conserving its energy unless a need arises otherwise. The figure first depicts the server registering itself to a Resource Directory over IP, and also supplies its alternative CoAP transport endpoints (in this case, SMS), in steps 1 and 2. The server can subsequently disable communication radio interfaces requiring greater energy (such as for IP-based comunication), powering it up sporadically for maintenance activities like registration renewals. At other times, it maintains communication in a low-power state by listening only for incoming SMS messages.

```
                        +---------+
                        |  CoRE   |
                   +-->|   RD    |<--+
                   | +-+---------+-+ |
                 3| |               | |1
                  | |4            2| |
                  | v               v |
+---------------+       +---------------+
|               |       |               |
|   CoAP    | UDP |       | UDP |   CoAP    |
| Client  +-----+  5    +-----+  Server |
|           | SMS +------>| SMS |         |
|           |     +<------+     |         |
+---------------+   6   +---------------+
```

Figure 3: CoAP client interacting with RD to discover a server's SMS-based endpoint

A CoAP client wishing to perform CoAP operations with an energy-constrained CoAP server may query a resource directory for the SMS-based endpoint of the server (steps 3 and 4). Subsequently, SMS-based CoAP communication can occur between the endpoints as shown by arrows 5 and 6. Alternatively, the incoming SMS can be also used by the server as a triggering event to temporarily power up its radio

interface so that UDP or other transport-based CoAP communication can instead be employed for low latency communication with the client.

3. Node Types based on Transport Availability

In [RFC7228], Tables 1, 3 and 4 introduced classification schemes for devices, in terms of their resource constraints, energy limitations and communication power. For this document, in addition to these capabilities, it seems useful to also identify devices based on their transport capabilities.

Name	Transport Availability
T0	Single transport
T1	Multiple transports, with one or more active at any point in time
T2	Multiple active and persistent transports at all times

Table 1: Classes of Available Transports

Type T0 nodes possess the capability of exactly 1 type of transport channel for CoAP, at all times. These include both active and sleepy nodes, which may choose to perform duty cycling for power saving.

Type T1 nodes possess multiple different transports, and can retrieve or expose CoAP resources over any or all of these transports. However, not all transports are constantly active and certain transport channels and interfaces could be kept in a mostly-off state for energy-efficiency, such as when using CoAP over SMS.

Type T2 nodes possess more than 1 transport, and multiple transports are simultaneously active at all times in a persistent manner. CoAP proxy nodes which allow CoAP endpoints from disparate transports to communicate with each other, are a good example of this.

4. New Resource Directory Parameters

In order to allow resource interactions between clients and servers with multiple locations or transports, the registration, update and lookup interfaces of the CoRE Resource Directory need to be extended. In this section two new RD parameters, "at" and "tt" are introduced. Both are optional CoAP features. If supported, they occur at the granularity level of an origin server, ie. they cannot be applied selectively on some resources only. When absent, it is assumed that the server does not support multiple transports or locations.

4.1. The 'at' RD parameter

A CoAP server wishing to advertise its resources over multiple transports does so by using one or more "at" parameters to register CoAP alternative transport URIs with a Resource Directory. Such a URI would contain the scheme, address as well as any port or paths at which the server is available.

Name	Query	Validity	Description	Value
CoAP Transport URI	at	URI	URI scheme, address port and path on the server	xsd:string

Table 2: The "at" RD parameter

The "at" parameter extends the Resource Directory's Registration and Update interfaces.

The following example shows a type T1 endpoint registering its resources and advertising its ability to use TCP and WebSockets as alternative transports:

```
Req: POST coap://rd.example.com/rd?ep=node1
   &at=coap+tcp://[2001:db8:f1::2]&at=coap+ws://server.example.com
Content-Format: 40
Payload:
</temperature>;ct=0;rt="temperature";if="core.s"

Res: 2.01 Created
Location: /rd/1234
```

An endpoint lookup would just reflect the registered attributes:

```
Req: GET /rd-lookup/ep

Res: 2.05 Content
</rd/1234>;ep="node1";base="coap://[2001:db8:f1::2]:5683";
   at="coap+tcp://[2001:db8:f1::2]";at="coap+ws://server.example.com"
```

The next example shows the same endpoint updating its registration with a new lifetime and the availability of a single alternative transport for CoAP (in this case TCP):

```
 Req: POST /rd/1234?lt=600
    &at=coap+tcp://[2001:db8:f1::2]
 Content-Format: 40
 Payload:
 </temperature>;ct=0;rt="temperature";if="core.s"

 Res: 2.04 Changed
```

If a lookup is performed on the same endpoint only 1 alternative transport is indicated:

```
 Req: GET /rd-lookup/ep

 Res: 2.05 Content
 </rd/1234>;ep="node1";base="coap://[2001:db8:f1::2]:5683";
    at="coap+tcp://[2001:db8:f1::2]"
```

A resource lookup for UDP client would be returned as the following:

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

Res: 2.05 Content
<coap://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";if="core.s";
   anchor="coap://[2001:db8:f1::2]"
```

A resource lookup for TCP client would be returned as the following:

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

Res: 2.05 Content
<coap+tcp://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";if="core.s";
   anchor="coap+tcp://[2001:db8:f1::2]"
```

4.2. The 'tt' RD parameter

A CoAP client wishing to perform a look-up on the Resource Directory for CoAP servers supporting multiple transports does so by using one or more "tt" parameters to query for CoAP alternative transport URIs.

Name	Query	Validity	Description	Value
CoAP Transport Type	tt		Transport type requested by the client	xsd:string

Table 3: The "tt" RD parameter

The "tt" parameter extends the Resource Directory's rd-lookup interface. The "tt" parameter queries existing registrations, and MUST NOT be used with the Resource Directory's registration and update interfaces.

The following example shows a client performing a lookup for endpoints supporting TCP:

```
Req: GET /rd-lookup/ep?tt="coap+tcp"

Res: 2.05 Content
</rd/1234>;at="coap+tcp://[2001:db8:f1::2]";ep="node1";ct="40"
```

The following example shows a client performing a resource lookup for endpoints supporting TCP:

```
Req: GET /rd-lookup/res?rt=temperature&tt="coap+tcp"

Res: 2.05 Content
<coap+tcp://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";
 if="core.s";anchor="coap+tcp://[2001:db8:f1::2]"
```

The following example shows a client performing a lookup for endpoints supporting SMS i.e. discovering SMS transports for sleepy nodes and using SMS to communicate with the endpoint:

```
Req: GET /rd-lookup/ep?et=oic.d.switch&tt="coap+sms"

Res: 2.05 Content
</rd/2345>;at="coap+sms://0015105550101/";ep="node5";
 et="oic.d.switch";ct="40",
</rd/4521>;at="coap+sms://0015105550202/";ep="node8";
 et="oic.d.switch";ct="40"
```

5. CoAP Alternative-Transport Option

The CoAP Alternative-Transport Option can be used by CoAP clients and CoAP servers in both Request and Response messages in constrained environments where a CoRE Resource Directory is not present.

Figure 4 depicts the properties of the Alternative-Transport Option.

No.	C	U	N	R	Name	Format	Length	Default
66		x	-	x	Alternative-Transport	string	0-1034	(none)

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

Figure 4: The Alternative-Transport Option

When included in a Request message, this option is used by the client in 2 possible ways. In the first case, a CoAP client can include the Option with Length 0 to retrieve all alternative transports from a CoAP server. In response to the client, the server includes base URI for each transport in its own Option. In the second case, a CoAP client can include the Option with a specific value in a CoAP Request, and the CoAP server returns the base URI(s) for the specified transport. If the specified transport by a CoAP client returns multiple results on a CoAP server, the server returns all base URIs of the transport in the response, each base URI in its own Option.

A CoAP client can also use this Option to retrieve several transports at once by including multiple Options in the request to a CoAP server. If any of the specified transports is supported by the

server, the server returns all base URIs in its own option. There can be more than 1 result for any of the transports so that each transport base URI is still included in the response in its own option.

Figure 5 describes a simple interaction between a client and a server, in which the client uses an Alternative-Transports Option with a null value to discover and retrieve all the available transports from the server, as part of a GET operation to retrieve a resource representation. The server responds with a CoAP Response message which contains the resource representation as a payload. In addition, the server also supplies multiple Alternative-Transport Options in the message, with each Option containing the base URI for an available transport. In this case the base URIs returned for TCP-based and WebSocket transports indicate their availability over a non-standard port.

```
Client                                    Server
   +                                        |
   |  GET /temperature                      |
   |  Token: 0x64                           |
   |  Alternative-Transport: (null)         |
   +---------------------------------------->|
   |                                        |
   |    2.05 Content                        |
   |    Token: 0x64                         |
   |    Payload: 21.0 Cel                   |
   |    Alternative-Transport:              |
   |      coap+tcp://example.org:5555/      |
   |    Alternative-Transport:              |
   |      coaps+tcp://example.org:6666/     |
   |    Alternative-Transport:              |
   |      coap+sms://0015105550101/         |
   |    Alternative-Transport:              |
   |      coap+ws://example.org:8080/       |
   |                                        |
   |<----------------------------------------+
   |                                        |
```

Figure 5: Requesting all available alternative transports on the server, and their locations

Alternatively, a client can also request for the availability of a specific transport on the server, as shown in Figure 6. Here, the CoAP Request contains Alternative-Transport Options with values set to request the Base URIs for TCP-based endpoints.

```
Client                                    Server
   +                                         |
   |  GET /temperature                       |
   |  Token: 0x64                            |
   |  Alternative-Transport: coap+tcp        |
   |  Alternative-Transport: coaps+tcp       |
   +---------------------------------------->|
   |                                         |
   |    2.05 Content                         |
   |    Token: 0x64                          |
   |    Payload: 21.0 Cel                    |
   |    Alternative-Transport:               |
   |      coap+tcp://example.org:5555/       |
   |    Alternative-Transport:               |
   |      coaps+tcp://example.org:6666/      |
   |                                         |
   |<----------------------------------------+
   |                                         |
```

Figure 6: Requesting TCP-based alternative transports on the server, and their locations

A client may also request a subset of available transports on the server, by providing multiple Options, each having a single transport identifier. The server likewise responds to the client request by supplying the requested transport information. This is shown in Figure 7.

```
              Client                                  Server
                 +                                       |
                 |  GET /temperature                     |
                 |  Token: 0x64                          |
                 |  Alternative-Transport:  ws           |
                 |  Alternative-Transport:  sms          |
                 +-------------------------------------->|
                 |                                       |
                 |    2.05 Content                       |
                 |    Token: 0x64                        |
                 |    Payload: 21.0 Cel                  |
                 |    Alternative-Transport:             |
                 |      coap+sms://0015105550101/        |
                 |    Alternative-Transport:             |
                 |      coap+ws://example.org:8080/      |
                 |                                       |
                 |<--------------------------------------+
                 |                                       |
```

Figure 7: Requesting WebSocket- and SMS-based alternative transports on the server, and their locations

6. The 'ol' CoRE Link Attribute

In the majority of cases, it is expected that an origin server would expose all its resources uniformly on its available transports or endpoint addresses. Exceptions can exist however, where alternate locations are made available on a per-resource basis. For such cases, a new 'ol' ("other locations") attribute is provided. One or more 'ol' attributes are used to provide base URIs from which a specific resource can be reached. Allowing per-resource endpoint or transport availability enables specific functions such as firmware updates or hardware-specific operations. It also facilitates mapping to and from OCF-based resource-specific endpoint descriptions. Note that the use of 'ol' is orthogonal to using 'at' as shown in Section 6.2.

6.1. Using /.well-known/core

```
REQ: GET /.well-known/core

RES: 2.05 Content
</sensors/temp>;ct=41;rt="temperature-f";if="sensor",
</sensors/door>;ct=41;rt="door";if="sensor",
</sensors/light>;if="sensor"; ol="http://[FDFD::123]:61616";
  ol="coap://server2.example.com"
```

6.2. Using CoRE Resource Directory

```
Req: POST coap:/rd.example.com/rd
 ?ep=node1&at=coap+tcp://server.example.com&at=coap+ws://server.example.com:5
683/ws/
Content-Format: 40
Payload:
</sensors/temp>;ct=41;rt="temperature-f";if="sensor",
</sensors/door>;ct=41;rt="door";if="sensor",
</sensors/light>;if="sensor"; ol="http://[FDFD::123]:61616";
  ol="coap://server2.example.com"

Res: 2.01 Created
Location: /rd/4521
```

7. IANA Considerations

This document requests the registration of new RD parameter types "at" and "tt".

The following entry needs to be added to the CoAP Option Numbers Registry:

Number	Name	Reference
66	Alternative-Transports	(this document)

8. Security Considerations

When multiple transports, locations and representations are used, some obvious risks are present both at the origin server as well as by requesting clients.

When a client is presented with alternate URIs for retrieving resources, it presents an opportunity for attackers to mount a series of attacks, either by hijacking communication and masquerading as an alternate location or by using a man-in-the-middle attack on TLS-based communication to a server and redirecting traffic to an alternate location. A malicious or compromised server could also be used for reflective denial-of-service attacks on innocent third parties. Moreover, clients may obtain web links to alternate URIs containing weaker security properties than the existing session.

9. Acknowledgements

Thanks to Christian Amsuess, Klaus Hartke, Jaime Jimenez and Jim Schaad for comments and reviewing this draft. Teemu Savolainen was involved in initial discussions about protocol negotations and lifetime values. Zach Shelby provided significant suggestions on how the Resource Directory can be employed and extended in place of link attributes and relation types.

10. References

10.1. Normative References

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-14 (work in progress), July 2018.

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

10.2. Informative References

[I-D.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-11 (work in progress), March 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[WWWArchv1] http://www.w3.org/TR/webarch/#uri-aliases, "Architecture of the World Wide Web, Volume One", December 2004.

Appendix A. Change Log

A.1. From -08 to -09

Using "tt" and "Alternative Transports" updated.

A.2. From -07 to -08

Added example of energy constrained CoAP server

Updated examples of using "at" and "tt"

"at" and "ol" are no longer comma-separated URI lists.

A.3. From -06 to -07

Added support for 'ol' Link attribute

A.4. From -05 to -06

Added support for CoAP Alternative-Transports Option

A.5. From -04 to -05

Freshness update

A.6. From -03 to -04

Removed previously introduced link attribute and relation types

Initial foray with Resource Directory support

A.7. From -02 to -03

Added new author

Rewrite of "Introduction" section

Added new Aims Section

Added new Section on Node Types

Introduced "al" Active Lifetime link attribute

Added new Section on Observing transports and resources

Security and IANA considerations sections populated

A.8. From -01 to -02

Freshness update.

A.9. From -00 to -01

Reworked "Introduction" section, added "Rationale", and "Goals" sections.

Authors' Addresses

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

Mert Ocak
Ericsson
Hirsalantie 11
02420 Jorvas
Finland

Email: mert.ocak@ericsson.com

CORE
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2016

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

November 03, 2015

A TCP and TLS Transport for the Constrained Application Protocol (CoAP)
draft-tschofenig-core-coap-tcp-tls-05

Abstract

The Hypertext Transfer Protocol (HTTP) was designed with TCP as the underlying transport protocol. The Constrained Application Protocol (CoAP), while inspired by HTTP, has been defined to make use of UDP instead of TCP. Therefore, reliable delivery and a simple congestion control and flow control mechanism are provided by the message layer of the CoAP protocol.

A number of environments benefit from the use of CoAP directly over a reliable byte stream such as TCP, which already provides these services. This document defines the use of CoAP over TCP as well as CoAP over TLS.

Status of This Memo

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP can be used unimpeded -- UDP [RFC0768], or DTLS [RFC6347] over UDP; it is a good choice for transferring small amounts of data across networks that follow the IP architecture. Some CoAP deployments, however, may have to integrate well with existing enterprise infrastructure, where the use of UDP-based protocols may not be well-received or may even be blocked by firewalls. Middleboxes that are unaware of CoAP usage

for IoT can make the use of UDP brittle, resulting in lost or malformed packets.

Where NATs are still present, CoAP over TCP can also help with their traversal. NATs often calculate expiration timers based on the transport layer protocol being used by application protocols. Many NATs are built around the assumption that a transport layer protocol such as TCP gives them additional information about the session life cycle and keep TCP-based NAT bindings around for a longer period. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter, as research confirms [HomeGateway].

Some environments may also benefit from the more sophisticated congestion control capabilities provided by many TCP implementations. (Note that there is ongoing work to add more elaborate congestion control to CoAP as well, see [I-D.bormann-core-cocoa].)

Finally, CoAP may be integrated into a Web environment where the front-end uses CoAP from IoT devices to a cloud infrastructure but the CoAP messages are then transported in TCP between the back-end services. A TCP-to-UDP gateway can be used at the cloud boundary to talk to the UDP-based IoT.

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

The present document describes a shim header that conveys length information about each CoAP message. Modifications to CoAP beyond the replacement of the message layer (e.g., to introduce further optimizations) are intentionally avoided.

2. Terminology

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

3. Constrained Application Protocol

The interaction model of CoAP over TCP is very similar to the one for CoAP over UDP, with the key difference that using TCP voids the need to provide certain transport layer protocol features, such as reliable delivery, fragmentation and reassembly, as well as congestion control, at the CoAP level. The protocol stack is illustrated in Figure 1 (derived from [RFC7252], Figure 1).

```
+----------------------+
|      Application     |
+----------------------+
+----------------------+
|  Requests/Responses  |  CoAP (RFC7252)
|----------------------|
|    Message adapter   |  This Document
+----------------------+
+-----------+    ^
|    TLS    |    |
+-----------+    v
+----------------------+
|          TCP         |
+----------------------+
```

Figure 1: The CoAP over TLS/TCP Protocol Stack

Since TCP offers reliable delivery, there is no need to offer a redundant acknowledgement at the CoAP messaging layer.

Since there is no need to carry around acknowledgement semantics, messages do not require a message type; no message layer acknowledgement is expected or even possible. Because something needs to be put into the two bits indicating the message type (unless alternative L3 below is chosen), we put the bits for a Non-Confirmable message (NON) into the header. By the nature of TCP, messages are always transmitted reliably over TCP. Figure 2 (derived from [RFC7252], Figure 3) shows this message exchange graphically. A UDP-to-TCP gateway will therefore discard all empty messages, such as empty ACKs (after operating on them at the message layer), and re-pack the contents of all non-empty CON, NON, or ACK messages (i.e., those ACK messages that have a piggy-backed response) into untyped messages (that happen to look like NON messages).

Similarly, there is no need to detect duplicate delivery of a message. In UDP CoAP, the Message ID is used for relating acknowledgements to Confirmable messages as well as for duplicate detection. Since the Message ID thus is not meaningful over TCP, it is elided (as indicated by the dashes in Figure 2).

```
Client                 Server
   |                      |
   | (no type) [------]   |
   +-------------------->|
   |                      |
```

Figure 2: Untyped Message Transmission over TCP.

A response is sent back as defined in [RFC7252], as illustrated in Figure 3 (derived from [RFC7252], Figure 6).

```
        Client                Server
           |                    |
           | (no type) [------] |
           | GET /temperature   |
           |   (Token 0x74)     |
           +------------------->|
           |                    |
           | (no type) [------] |
           |   2.05 Content     |
           |   (Token 0x74)     |
           |     "22.5 C"       |
           |<-------------------+
           |                    |
```

Figure 3

4. Message Format

The CoAP message format defined in [RFC7252], as shown in Figure 4, relies on the datagram transport (UDP, or DTLS over UDP) for keeping the individual messages separate.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: RFC 7252 defined CoAP Message Format.

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Message Length         |Ver|0 1|  TKL  |      Code     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Token (TKL bytes) ...  |  Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: CoAP Header with prepended Shim Header.

The 'Message Length' field is a 16-bit unsigned integer in network byte order. It provides the length of the subsequent CoAP message (including the CoAP header but excluding this message length field) in bytes (so its minimum value is 2). The Message ID and message type are meaningless and thus elided (what would have been the message type field is always filled with what would be the code for NON (01)).

The semantics of the other CoAP header fields are left unchanged.

4.1. Discussion

One observation is that 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 larger than 1024 bytes, potentially also obviating the need for the Block protocol [I-D.ietf-core-block]. It must be noted that entirely getting rid of the block protocol is not a generally applicable solution, as:

- o a UDP-to-TCP gateway may simply not have the context to convert a message with a Block option into the equivalent exchange without any use of a Block option.

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

The general assumption is therefore that the block protocol will continue to be used over TCP, even if TCP-based applications occasionally do exchange messages with payload sizes larger than desirable in UDP.

5. Message Transmission

As CoAP exchanges messages asynchronously over the TCP connection, the client can send multiple requests without waiting for responses. For this reason, and due to the nature of TCP, responses are returned

during the same TCP connection as the request. In the event that the connection gets terminated, all requests that have not yet elicited a response are implicitly canceled; clients may transmit the request again once a connection is reestablished.

Furthermore, since TCP is bidirectional, requests can be sent from both the connecting host and the endpoint that accepted the connection. In other words, whoever initiated the TCP connection has no bearing on the meaning of the CoAP terms client and server.

6. CoAP URI

CoAP [RFC7252] defines the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. RFC 7252 defines these resources for use with CoAP over UDP.

The present specification introduces two new URI schemes, namely "coap+tcp" and "coaps+tcp". The rules from Section 6 of [RFC7252] apply to these two new URI schemes.

[RFC7252], Section 8 (Multicast CoAP), does not apply to the URI schemes defined in the present specification.

Resources made available via one of the "coap+tcp" or "coaps+tcp" schemes have no shared identity with the other scheme or with the "coap" or "coaps" scheme, even if their resource identifiers indicate the same authority (the same host listening to the same port). The schemes constitute distinct namespaces and, in combination with the authority, are considered to be distinct origin servers.

6.1. coap+tcp URI scheme

```
coap-tcp-URI = "coap+tcp:" "//" host [ ":" port ] path-abempty
               [ "?" query ]
```

The semantics defined in [RFC7252], Section 6.1, apply to this URI scheme, with the following changes:

- o The port subcomponent indicates the TCP port at which the CoAP server is located. (If it is empty or not given, then the default port 5683 is assumed, as with UDP.)

6.2. coaps+tcp URI scheme

```
coaps-tcp-URI = "coaps+tcp:" "//" host [ ":" port ] path-abempty
                [ "?" query ]
```

The semantics defined in [RFC7252], Section 6.2, apply to this URI scheme, with the following changes:

- o The port subcomponent indicates the TCP port at which the TLS server for the CoAP server is located. If it is empty or not given, then the default port 443 is assumed (this is different from the default port for "coaps", i.e., CoAP over DTLS over UDP).

- o When CoAP is exchanged over TLS port 443 then the "TLS Application Layer Protocol Negotiation Extension" [RFC7301] MUST be used to allow demultiplexing at the server-side unless out-of-band information ensures that the client only interacts with a server that is able to demultiplex CoAP messages over port 443. This would, for example, be true for many IoT deployments where clients are pre-configured to only ever talk with specific servers. [[_1: Shouldn't we simply always require ALPN? The protocol should not be defined in such a way that it depends on some undefined pre-configuration mechanism. --cabo]]

7. Security Considerations

This document defines how to convey CoAP over TCP and TLS. It does not introduce new vulnerabilities beyond those described already in the CoAP specification. CoAP [RFC7252] makes use of DTLS 1.2 and this specification consequently uses TLS 1.2 [RFC5246]. CoAP MUST NOT be used with older versions of TLS. Guidelines for use of cipher suites and TLS extensions can be found in [I-D.ietf-dice-profile].

8. IANA Considerations

8.1. Service Name and Port Number Registration

IANA is requested to assign the port number 5683 and the service name "coap+tcp", in accordance with [RFC6335].

Service Name.
   coap+tcp

Transport Protocol.
   tcp

Assignee.
   IESG <iesg@ietf.org>

Contact.
   IETF Chair <chair@ietf.org>

Description.

Constrained Application Protocol (CoAP)

Reference.
[RFCthis]

Port Number.
5683

Similarly, IANA is requested to assign the service name "coaps+tcp", in accordance with [RFC6335]. However, no separate port number is used for "coaps" over TCP; instead, the ALPN protocol ID defined in Section 8.3 is used over port 443.

Service Name.
coaps+tcp

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFC7301], [RFCthis]

Port Number.
443 (see also Section 8.3 of [RFCthis]})

8.2. URI Schemes

This document registers two new URI schemes, namely "coap+tcp" and "coaps+tcp", for the use of CoAP over TCP and for CoAP over TLS over TCP, respectively. The "coap+tcp" and "coaps+tcp" URI schemes can thus be compared to the "http" and "https" URI schemes.

The syntax of the "coap" and "coaps" URI schemes is specified in Section 6 of [RFC7252] and the present document re-uses their semantics for "coap+tcp" and "coaps+tcp", respectively, with the exception that TCP, or TLS over TCP is used as a transport protocol.

IANA is requested to add these new URI schemes to the registry established with [RFC7595].

8.3. ALPN Protocol ID

This document requests a value from the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" created by [RFC7301]:

Protocol:
CoAP

Identification Sequence:
0x63 0x6f 0x61 0x70 ("coap")

Reference:
[RFCthis]

9. Acknowledgements

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Michael Koster, Matthias Kovatsch, Szymon Sasin, Andrew Summers, and Zach Shelby for their feedback.

10. References

10.1. Normative References

[I-D.ietf-dice-profile]
Tschofenig, H. and T. Fossati, "TLS/DTLS Profiles for the Internet of Things", draft-ietf-dice-profile-17 (work in progress), October 2015.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

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

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

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

10.2. Informative References

[HomeGateway]
Eggert, L., "An experimental study of home gateway characteristics", Proceedings of the 10th annual conference on Internet measurement, 2010.

[I-D.bormann-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-bormann-core-cocoa-03 (work in progress), October 2015.

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <http://www.rfc-editor.org/info/rfc768>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <http://www.rfc-editor.org/info/rfc6335>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

Authors' Addresses

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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

Simon Lemay
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: slemay@zebra.com

Valik Solorzano Barboza
Zebra Technologies
820 W. Jackson Blvd. suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: vsolorzanobarboza@zebra.com

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

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

CoRE
Internet-Draft
Intended status: Standards Track
Expires: July 22, 2017

P. van der Stok
consultant
A. Bierman
YumaWorks
M. Veillette
Trilliant Networks Inc.
A. Pelov
Acklio

January 18, 2017

CoAP Management Interface
draft-vanderstok-core-comi-11

Abstract

This document describes a network management interface for constrained devices and networks, called CoAP Management Interface (CoMI). The Constrained Application Protocol (CoAP) is used to access data resources specified in YANG, or SMIv2 converted to YANG. CoMI uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction. CoMI extends the set of YANG based protocols, NETCONF and RESTCONF, with the capability to manage constrained devices and networks.

Note

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

This draft describes the CoAP Management Interface which uses CoAP methods to access structured data defined in YANG [RFC7950]. This draft is complementary to the draft [I-D.ietf-netconf-restconf] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data sets, specified in a standardized language such as YANG, promotes interoperability between devices and applications from different manufacturers. A large amount of Management Information Base (MIB) [mibreg] specifications already exists for monitoring purposes. This data can be accessed in RESTCONF or CoMI if the server converts the SMIv2 modules to YANG, using the mapping rules defined in [RFC6643].

CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips.

To promote small packets, CoMI uses a YANG to CBOR mapping [I-D.ietf-core-yang-cbor] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length.

1.1. Terminology

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

Readers of this specification should be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

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

The following terms are defined in the YANG data modelling language [RFC7950]: anydata, anyxml, container, data node, key, key leaf, leaf, leaf-list, and list.

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

The following terms are defined in this document:

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

Notification instance: An instance of a schema node of type notification, specified in a YANG module present in the server. The instance is generated in the server at the occurrence of the corresponding event and appended to a stream.

YANG schema item identifier: Numeric identifier which replaces the name identifying a YANG item (see section 6.2 of [RFC7950]) (anydata, anyxml, data node, RPC, Action, Notification, Identity, Module name, Submodule name, Feature).

list instance identifier: Handle used to identify a YANG data node that is an instance of a YANG "list" specified with the values of the key leaves of the list.

single instance identifier: Handle used to identify a specific data node which can be instantiated only once. This includes data nodes defined at the root of a YANG module or submodule and data

nodes defined within a container. This excludes data nodes defined within a list or any children of these data nodes.

instance identifier: List instance identifier or single instance identifier.

data node value: Value assigned to a data node instance. Data node values are encoded based on the rules defined in section 4 of [I-D.ietf-core-yang-cbor].

set of data node instances: Represents the payload of CoAP methods when a collection is sent or returned. There are two possibilities, dependent on Request context:

1. CBOR array of pair(s) <instance identifier, data node value >

2. CBOR map of pair(s) <instance identifier, data node value >

TODO: Reduce to one, if possible

The following list contains the abbreviations used in this document.

SID: YANG Schema Item iDentifier.

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for the reading and modifying the content of a datastore used for the management of the instrumented node.

```
   +-------------------------------------------------------------+
   |                    SMIv2  specification (2)                 |
   +-------------------------------------------------------------+
                                \/
   +-------------------------------------------------------------+
   |                    YANG  specification  (1)                 |
   +---------*-----------------------------------------*----------+
             | compilation                             | compilation
             |               security (7)              |
   client   \|/      [==========================]    \|/    Server
   +--------------------+                       +------------------+
   |                    |                       |                  |
   | Request generation +--> CoAP request(3) -->| Request retrieval|
   | answer retrieval   |<-- CoAP response(3)<--| answer generation|
   |       (4)          |                       |       (4)        |
   +--------------------+                       | +---------------+|
                                                | | datastore (5) |
                                                | +---------------+|
                                                |                  |
                                                |     Variable     |
                                                |Instrumentation(6)|
                                                +------------------+
```

Figure 1: Abstract CoMI architecture

Figure 1 is a high level representation of the main elements of the CoAP management architecture. A client sends requests as payload in packets over the network to a managed constrained node.

The different numbered components of Figure 1 are discussed according to component number.

(1) YANG specification: contains a set of named and versioned modules.

(2) SMIv2 specification: A named module specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIv2 specifications to YANG specifications.

(3) CoMI messages: The CoMI client sends request messages to and receives response messages from the CoMI server.

(4) Retrieval, generation: The server and client parse the CoMI request/response and identify the corresponding instances in the datastore based on YANG specification.

(5) Datastore: The store is composed of two parts: Operational state and Configuration datastore. Datastore also supports RPCs and event streams.

(6) Variable instrumentation: This code depends on implementation of drivers and other node specific aspects.

(7) Security: The server MUST prevent unauthorized users from reading or writing any data resources. CoMI relies on security protocols such as DTLS [RFC6347] to secure CoAP communication.

2.1. Major differences between RESTCONF and CoMI

CoMI uses CoAP/UDP as transport protocol and CBOR as payload format [I-D.ietf-core-yang-cbor]. RESTCONF uses HTTP/TCP as transport protocol and JSON [RFC7159] or XML [XML] as payload formats. CoMI encodes YANG identifier strings as numbers, where RESTCONF does not.

CoMI uses the methods FETCH and iPATCH, not used by RESTCONF. RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not used by CoMI.

CoMI servers cannot change the order of user-ordered data. CoMI does not support insert-mode (first, last, before, after) and insertion-point (before, after) which are supported by RESTCONF. Many CoAP servers will not support date and time functions. For that reason CoMI does not support the start, stop options for events.

CoMI servers only implement the efficient "trim" mode for default values.

The CoMI servers do not support the following RESTCONF functionality:

- The "fields" query parameter to query multiple instances.

- The 'filter' query that involves XML parsing, 'content', and 'depth', query parameters.

2.2. Compression of YANG identifiers

In the YANG specification items are identified with a name string. In order to significantly reduce the size of identifiers used in CoMI, numeric object identifiers are used instead of these strings. The specific encoding of the object identifiers is not hard-wired in the protocol.

Examples of object identifier encoding formats are described in [I-D.ietf-core-sid].

3. Example syntax

This section presents the notation used for the examples. The YANG specifications that are used throughout this document are shown in Appendix A. The example specifications are taken over from existing modules and annotated with SIDs. The values of the SIDs are taken over from [yang-cbor].

CBOR is used to encode CoMI request- and response- payloads. The CBOR syntax of the YANG payloads is specified in [RFC7049]. The payload examples are notated in Diagnostic notation (defined in section 6 of [RFC7049]) that can be automatically converted to CBOR.

A YANG (item identifier, item value) pair is mapped to a CBOR (key, value) pair. The YANG item value is encoded as specified in [I-D.ietf-core-yang-cbor]. The YANG item identifier can be a SID (single node identifier) or a CBOR array with the structure [SID, key1, key2] (list node identifier), where SID is a list identifier and the key values specify the list instance. The YANG item value can be any CBOR major type.

Delta encoding is used for the SIDs. The notation +n is used when the SID has the value PREC+n where PREC is the SID of the parent container, or PREC is the SID of the preceding entity in a CBOR array.

In all examples the resource path in the URI is expressed as a SID, represented as a base64 number. SIDs in the payload are represented as decimal numbers.

4. CoAP Interface

In CoAP a group of links can constitute a Function Set.

TODO: what will happen to term Function Set ?

The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Function Set. CoMI end-points that implement the CoMI management protocol support at least one discoverable management resource of resource type (rt): core.c, with path: /c, where c is short-hand for CoMI. The path root /c is recommended but not compulsory (see Section 8).

The path prefix /c has resources accessible with the following three paths:

/c: YANG-based data with path "/c" and using CBOR content encoding format. This path represents a datastore resource which contains

YANG data resources as its descendant nodes. The data nodes are identified with their SID with format /c/SID.

/c/mod.uri: URI identifying the location of the server module information, with path "/c/mod.uri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG encoded values. An Entity Tag MUST be maintained for this resource by the server, which MUST be changed to a new value when the set of YANG modules in use by the server changes.

/c/s: String identifying the default stream resource to which YANG notification instances are appended. Notification support is optional, so this resource will not exist if the server does not support any notifications.

The mapping of YANG data node instances to CoMI resources is as follows: A YANG module describes a set of data trees composed of YANG data nodes. Every data node of the YANG modules loaded in the CoMI server represents a resource of the datastore container (e.g. /c/<sid>

When multiple instances of a list node exist, instance selection is possible as described in Section 5.2.4 and Section 5.2.3.1.

TODO; reference to fetch and patch content formats.

The profile of the management function set, with IF=core.c, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

name	path	rt	Data Type
Management	/c	core.c	n/a
Data	/c	core.c.data	application/cbor
Module Set URI	/c/mod.uri	core.c.moduri	application/cbor
Events	/c/s	core.c.stream	application/cbor

5. /c Function Set

The /c Function Set provides a CoAP interface to manage YANG servers.

The methods used by CoMI are:

Operation	Description
GET	Retrieve the datastore resource or a data resource
FETCH	Retrieve (partial) data resource(s)
POST	Create a data resource, invoke RPC
PUT	Create or replace a data resource
iPATCH	Idem-potently create, replace, and delete data resource(s) (partially)
DELETE	Delete a data resource

There is one query parameters for the GET, PUT, POST, and DELETE methods.

Query Parameter	Description
k	Select an instance of a list node

This parameter is not used for FETCH and iPATCH, because their request payloads support list instance selection.

5.1. Using the 'k' query parameter

The "k" (key) parameter specifies the instance of a list node. The SID in the URI is followed by the (?k=key1, key2,..). Where SID identifies a list node, and key1, key2 are the values of the key leaves that specify an instance of the list.

Key values are encoded using the rules defined in the following table:

YANG datatype	Binary representation	Text representation
uint8,uint16,unit32, uint64	CBOR unsigned integer	int_to_text(number)
int8, int16,int32, int64	CBOR negative integer	base64 (CBOR representation)
decimal64	CBOR decimal fractions	base64 (CBOR representation
string	CBOR text or string	text
boolean	CBOR false or true	"0" or "1"
enumeration	CBOR unsigned integer	int_to_text (number)
bits	CBOR byte string	base64 (CBOR representation)
binary	CBOR byte string	base64 (binary value)
identityref	CBOR unsigned integer	int_to_text (number)
union		base64 (CBOR representation)
List instance identifier	CBOR unsigned integer	base64 (CBOR representation)
List instance identifier	CBOR array	Base64 (CBOR representation)

5.2. Data Retrieval

One or more data node instances can be retrieved by the client. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252] and to the FETCH method defined in section 2 of [I-D.ietf-core-etch].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [RFC7959] is used, as explained in more detail in Section 7.

CoMI uses the FETCH payload for retrieving a subset of the datastore.

There are two additional query parameters for the GET and FETCH methods.

Query Parameter	Description
c	Control selection of configuration and non-configuration data nodes (GET and FETCH)
d	Control retrieval of default values.

5.2.1. Using the 'c' query parameter

The 'c' (content) parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
c	Return only configuration descendant data nodes
n	Return only non-configuration descendant data nodes
a	Return all descendant data nodes

This parameter is only allowed for GET and FETCH methods on datastore and data resources. A 4.00 Bad Request error is returned if used for other methods or resource types.

If this query parameter is not present, the default value is "a".

5.2.2. Using the 'd' query parameter

The "d" (with-defaults) parameter controls how the default values of the descendant nodes of the requested data nodes will be processed.

The allowed values are:

Value	Description
a	All data nodes are reported Defined as 'report-all' in section 3.1 of [RFC6243].
t	Data nodes set to the YANG default are not reported. Defined as 'trim' in section 3.2 of [RFC6243].

If the target of a GET or FETCH method is a data node that represents a leaf that has a default value, and the leaf has not been given a value yet, the server MUST return the leaf.

If the target of a GET method is a data node that represents a container or list that has any child resources with default values, for the child resources that have not been given value yet, the server MUST not return the child resource if this query parameter is set to 't' and MUST return the child resource if this query parameter is set to 'a'.

If this query parameter is not present, the default value is 't'.

5.2.3. GET

A request to read the values of a data node instance is sent with a confirmable CoAP GET message. A single instance identifier is specified in the URI path prefixed with /c.

FORMAT:

```
GET /c/<instance identifier>

2.05 Content (Content-Format: application/cbor)
<data node value>
```

The returned payload is composed of all the children associated with the specified data node instance.

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.2.3.1. GET Examples

Using for example the current-datetime leaf from Appendix A.1, a request is sent to retrieve the value of system-state/clock/current-datetime specified in container system-state. The ID of system-

state/clock/current-datetime is 1719, encoded in base64 this yields a3. The answer to the request returns a <value>, transported as a single CBOR string item.

```
REQ: GET example.com/c/a3

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

For example, the GET of the clock node (ID = 1717; base64: a1), sent by the client, results in the following returned value sent by the server, transported as a CBOR map containing 2 pairs:

```
REQ: GET example.com/c/a1

RES: 2.05 Content (Content-Format: application/cbor)
{
      +2 : "2014-10-26T12:16:51Z",   / ID 1719 /
      +1 : "2014-10-21T03:00:00Z"    / ID 1718 /
}
```

A "list" node can have multiple instances. Accordingly, the returned payload of GET is composed of all the instances associated with the selected list node.

For example, look at the example in Appendix A.3. The GET of the /interfaces/interface/ (with identifier 1533, base64: X9) results in the following returned payload, transported as a CBOR array with 2 elements.

```
REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/cbor)
[
 {+4 : "eth0",                 / name  (ID 1537) /
  +1 : "Ethernet adaptor",     / description (ID 1534) /
  +5 : 1179,                   / type, (ID 1538) identity /
                               / ethernetCsmacd (ID 1179) /
  +2 : true                    / enabled ( ID 1535) /
 },
 {+4 : "eth1",                 / name (ID 1537) /
  +1 : "Ethernet adaptor",     / description (ID 1534) /
  +5 : 1179,                   / type, (ID 1538) identity /
                               / ethernetCsmacd (ID 1179) /
  +2 : false                   / enabled /
  }
]
```

It is equally possible to select a leaf of one instance of a list or a complete instance container with GET. The instance identifier is the numeric identifier of the list followed by the specification of the values for the key leaves that uniquely identify the list instance. The instance identifier looks like: SID?k=key-value. The key of "interface" is the "name" leaf. The example below requests the description leaf of the instance with name="eth0" (ID=1534, base64: X-). The value of the description leaf is returned.

```
REQ: GET example.com/c/X-?k="eth0"

RES: 2.05 Content (Content-Format: application/cbor)
"Ethernet adaptor"
```

5.2.4. FETCH

The FETCH is used to retrieve a list of data node values. The FETCH Request payload contains a CBOR list of instance identifiers.

```
FORMAT:
    FETCH /c/ Content-Format (application/YANG-fetch+cbor)
    <CBOR array of instance identifiers>

    2.05 Content (Content-Format: application/YANG-patch+cbor)
    <CBOR array of data node values>
```

The instance identifier is a SID or a CBOR array containing the SID followed by key values that identify the list instance (sec 5.13.1 of [I-D.ietf-core-yang-cbor]. In the payload of the returned data node

values, delta encoding is used as described in [I-D.ietf-core-yang-cbor].

5.2.4.1. FETCH examples

The example uses the current-datetime leaf and the interface list from Appendix A.1. In the following example the value of current-datetime (ID 1719)and the interface list (ID 1533) instance identified with name="eth0" are queried.

```
REQ:  FETCH /c Content-Format (application/YANG-fetch+cbor)
      [ 1719,            / ID 1719 /
      [-186, "eth0"]   / ID 1533 with name = "eth0" /
      ]

RES:  2.05 Content Content-Format (application/YANG-patch+cbor)
[
  "2014-10-26T12:16:31Z",
 {
   +4 : "eth0",                / name (ID 1537) /
   +1 : "Ethernet adaptor",    / description (ID 1534) /
   +5 : 1179,                  / type (ID 1538), identity /
                               / ethernetCsmacd (ID 1179) /
   +2 : true                   / enabled (ID 1535) /
  }
]
```

TODO: align with future FETCH content format.

5.3. Data Editing

CoMI allows datastore contents to be created, modified and deleted using CoAP methods.

5.3.1. Data Ordering

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as arrays so messages will preserve their order.

5.3.2. POST

Data resources are created with the POST method. The CoAP POST operation is used in CoMI for creation of data resources and the invocation of "ACTION" and "RPC" resources. Refer to Section 5.6 for details on "ACTION" and "RPC" resources.

A request to create the values of an instance of a container or leaf is sent with a confirmable CoAP POST message. A single SID is specified in the URI path prefixed with /c.

```
FORMAT:
    POST /c/<instance identifier> Content-Format(application/cbor)
    <data node value>

    2.01 Created (Content-Format: application/cbor)
```

If the data resource already exists, then the POST request MUST fail and a "4.09 Conflict" status-line MUST be returned

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.2.1. Post example

The example uses the interface list from Appendix A.1. Example is creating a new version of the container interface (ID = 1533):

```
REQ: POST /c/X9 Content-Format(application/cbor)
      {
        +4 : "eth0",              / name (ID 1537) /
        +1 : "Ethernet adaptor", / description (ID 1534) /
        +5 : 1179,                / type (ID 1538), identity /
                                  / ethernetCsmacd (ID 1179) /
        +2 : true                 / enabled (ID 1535) /
      }

RES: 2.01 Created (Content-Format: application/cbor)
```

5.3.3. PUT

Data resource instances are created or replaced with the PUT method. The PUT operation is supported in CoMI. A request to set the value of a data node instance is sent with a confirmable CoAP PUT message.

```
FORMAT:
    PUT /c/<instance identifier> Content-Format(application/cbor)
    <data node value>

    2.01 Created
```

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.3.1. PUT example

The example uses the interface list from Appendix A.1. Example is renewing an instance of the list interface (ID = 1533) with key name="eth0":

```
REQ:  PUT /c/X9?k="eth0" Content-Format(application/cbor)
      {
        +4 : "eth0",             / name (ID 1537) /
        +1 : "Ethernet adaptor", / description (ID 1534) /
        +5 : 1179,               / type (ID 1538), identity /
                                 / ethernetCsmacd ( ID 1179) /
        +2 : true                / enabled (ID 1535) /
      }
RES:  2.04 Changed
```

5.3.4. iPATCH

One or multiple data resource instances are replaced with the idempotent iPATCH method [I-D.ietf-core-etch]. A request is sent with a confirmable CoAP iPATCH message.

There are no query parameters for the iPATCH method.

The processing of the iPATCH command is specified by the CBOR payload. The CBOR patch payload describes the changes to be made to target YANG data nodes [I-D.bormann-appsawg-cbor-merge-patch]. If the CBOR patch payload contains data node instances that are not present in the target, these instances are added or silently ignored dependent of the payload information. If the target contains the specified instance, the contents of the instances are replaced with the values of the payload. Null values indicate the removal of existing values.

```
FORMAT:
    iPATCH /c Content-Format(application/YANG-patch+cbor)
    <set of data node instances>

    2.04 Changed
```

5.3.4.1. iPATCH example

The example uses the interface list from Appendix A.3, and the timezone-utc-offset leaf from Appendix A.1. In the example one leaf (timezone-utc-offset) and one container (interface) instance are changed.

```
REQ: iPATCH /c Content-Format(application/YANG-patch+cbor)
[
  [1533, "eth0"] ,                / interface (ID = 1533) /
    {
      +4 : "eth0",                / name (ID 1537) /
      +1 : "Ethernet adaptor",    / description (ID 1534) /
      +5 : 1179,                  / type (ID 1538), identity /
                                  / ethernetCsmacd (ID 1179) /
      +2 : true                   / enabled (ID 1535) /
    },
  +203 , 60          / timezone-utc-offset (delta = 1736-1533) /
]

RES: 2.04 Changed

TODO: Align with future cbor-merge-patch content format.
```

5.3.5. DELETE

Data resource instances are deleted with the DELETE method. The RESTCONF DELETE operation is supported in CoMI.

```
FORMAT:
    Delete /c/<instance identifier>

    2.02 Deleted
```

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.5.1. DELETE example

The example uses the interface list from Appendix A.3. Example is deleting an instance of the container interface (ID = 1533):

```
REQ:   DELETE /c/X9?k="eth0"

RES:   2.02 Deleted
```

5.4. Full Data Store access

The methods GET, PUT, POST, and DELETE can be used to return, replace, create, and delete the whole data store respectively.

```
FORMAT:
    GET /c
    2.05 Content (Content-Format: application/cbor)
    <array of data node instances>

   PUT /c
   (Content-Format: application/cbor)
    <array of data node instances>
   2.04 Changed

   POST /c
   (Content-Format: application/cbor)
    <array of data node instances>
   2.01 Created

   DELETE /c
   2.02 Deleted
```

The array of data node instances represents an array of all root nodes in the data store after the PUT, POST and GET method invocations.

5.4.1. Full Data Store examples

The example uses the interface list and the clock container from Appendix A.3. Assume that the data store contains two root objects: the list interface (ID 1533) with one instance and the container Clock (ID 1717). After invocation of GET an array with these two objects is returned:

```
RQ:  GET /c
RES: 2.05 Content Content-Format (application/YANG-patch+cbor)
[
  {1717:
     { +2: "2016-10-26T12:16:31Z", / current-datetime (ID 1719) /
       +1: "2014-10-05T09:00:00Z"  / boot-datetime (ID 1718) /
     },
  -186:                            / clock (ID 1533) /
     {
       +4 : "eth0",                / name (ID 1537) /
       +1 : "Ethernet adaptor",    / description (ID 1534) /
       +5 : 1179,                  / type (ID 1538), identity: /
                                   / ethernetCsmacd (ID 1179) /
       +2 : true                   / enabled (ID 1535) /
      }
    }
]
```

5.5. Notify functions

Notification by the server to a selection of clients when an event occurs in the server is an essential function for the management of servers. CoMI allows events specified in YANG [RFC5277] to be notified to a selection of requesting clients. The server appends newly generated events to a stream. There is one, so-called "default", stream in a CoMI server. The /c/s resource identifies the default stream. The server MAY create additional stream resources. When a CoMI server generates an internal event, it is appended to the chosen stream, and the content of a notification instance is ready to be sent to all CoMI clients which observe the chosen stream resource.

Reception of generated notification instances is enabled with the CoAP Observe [RFC7641] function. The client subscribes to the notifications by sending a GET request with an "Observe" option, specifying the /c/s resource when the default stream is selected.

Every time an event is generated, the chosen stream is cleared, and the generated notification instance is appended to the chosen stream(s). After appending the instance, the contents of the instance is sent to all clients observing the modified stream.

```
FORMAT:
  Get /<stream-resource>
      Content-Format(application/YANG-patch+cbor) Observe(0)

2.05 Content Content-Format(application/YANG-patch+cbor)
<set of data node instances>
```

5.5.1. Notify Examples

Suppose the server generates the event specified in Appendix A.4. By executing a GET on the /c/s resource the client receives the following response:

```
REQ:  GET /c/s Observe(0) Token(0x93)

RES:  2.05 Content Content-Format(application/YANG-patch+cbor)
                        Observe(12) Token(0x93)
{
   60010 :                  / example-port-fault (ID 60010) /
    {
      +1 : "0/4/21",       / port-name (ID 60011) /
      +2 : "Open pin 2"    / port-fault (ID 60012) /
    },
   60010 :                  / example-port-fault (ID 60010) /
    {
      +1 : "1/4/21",       / port-name (ID 60011) /
      +2 : "Open pin 5"    / port-fault (ID 60012) /
    }
}
```

In the example, the request returns a success response with the contents of the last two generated events. Consecutively the server will regularly notify the client when a new event is generated.

To check that the client is still alive, the server MUST send confirmable notifications once in a while. When the client does not confirm the notification from the server, the server will remove the client from the list of observers [RFC7641].

5.6. RPC statements

The YANG "action" and "RPC" statements specify the execution of a Remote procedure Call (RPC) in the server. It is invoked using a POST method to an "Action" or "RPC" resource instance. The Request payload contains the values assigned to the input container when specified with the action station. The Response payload contains the values of the output container when specified.

The returned success response code is 2.05 Content.

```
FORMAT:
 POST /c/<instance identifier>
           Content-Format(application/YANG-patch+cbor)
<input node value>

2.05 Content Content-Format (application/YANG-patch+cbor)
<output node value>
```

There "k" query parameter is allowed for the POST method when used for an action invocation.

5.6.1. RPC Example

The example is based on the YANG action specification of Appendix A.2. A server list is specified and the action "reset" (ID 60002, base64: Opq), that is part of a "server instance" with key value "myserver", is invoked.

```
REQ:  POST /c/Opq?k="myserver"
              Content-Format(application/YANG-patch+cbor)
{
  +1 : "2016-02-08T14:10:08Z09:00" / reset-at (ID 60003) /
}

RES:  2.05 Content Content-Format(application/YANG-patch+cbor)
{
  +2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (ID 60004)/
}
```

6. Access to MIB Data

Appendix A.5 shows a YANG module mapped from the SMI specification "IP-MIB" [RFC4293]. The following example shows the "ipNetToPhysicalEntry" list with 2 instances, using diagnostic notation without delta encoding.

```
{
   60021 :                      / list ipNetToPhysicalEntry /
   [
     {
       60022 : 1,               / ipNetToPhysicalIfIndex /
       60023 : 1,               / ipNetToPhysicalNetAddressType: ipv4 /
       60024 : h'0A000033',     / ipNetToPhysicalNetAddress /
       60025 : h'00000A01172D',/ ipNetToPhysicalPhysAddress /
       60026 : 2333943,         / ipNetToPhysicalLastUpdated /
       60027 : 4,               / ipNetToPhysicalType: static /
       60028 : 1,               / ipNetToPhysicalState: reachable /
       60029 : 1                / ipNetToPhysicalRowStatus: active /
     },
     {
       60022 : 1,               / ipNetToPhysicalIfIndex /
       60023 : 1,               / ipNetToPhysicalNetAddressType: ipv4 /
       60024 : h'09020304',     / ipNetToPhysicalNetAddress  /
       60025 : h'00000A36200A',/ ipNetToPhysicalPhysAddress /
       60026 : 2329836,         / ipNetToPhysicalLastUpdated /
       60027 : 3,               / ipNetToPhysicalType: dynamic /
       60028 : 6,               / ipNetToPhysicalState: unknown /
       60029 : 1                / ipNetToPhysicalRowStatus: active /
     }
   ]
}
```

The IPv4 addresses A.0.0.33 and 9.2.3.4 are encoded in CBOR as h'0A000033' and h'09020304' respectively. In the following example exactly one instance is requested from the ipNetToPhysicalEntry (ID 60021, base64: Oz1). The h'09020304' value is encoded in base64 as AJAgME.

In this example one instance of /ip/ipNetToPhysicalEntry that matches the keys ipNetToPhysicalIfIndex = 1, ipNetToPhysicalNetAddressType = ipv4 and ipNetToPhysicalNetAddress = 9.2.3.4 (h'09020304', base64: AJAgME).

```
REQ: GET example.com/c/Oz1?k="1,1,AJAgME"

RES: 2.05 Content (Content-Format: application/YANG-patch+cbor)
{
   +1 : 1,                  / ( SID 60022 ) /
   +2 : 1,                  / ( SID 60023 ) /
   +3 : h'09020304',        / ( SID 60024 ) /
   +4 : h'00000A36200A',    / ( SID 60025 ) /
   +5 : 2329836,            / ( SID 60026 ) /
   +6 : 3,                  / ( SID 60027 ) /
   +7 : 6,                  / ( SID 60028 ) /
   +8 : 1                   / ( SID 60029 ) /
}
```

7. Use of Block

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of text need to be transported the datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [RFC7959] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying transport sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

Beware of race conditions. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. In the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the variables, the transported results do not correspond with a state having occurred in the server; or worse the returned values are inconsistent. For example: array length does not correspond with actual number of items. It may be advisable to use CBOR maps or CBOR arrays of undefined length which are foreseen for data streaming purposes.

8. Resource Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c" [RFC6690].

Upon success, the return payload will contain the root resource of the management data. It is up to the implementation to choose its root resource, but it is recommended that the value "/c" is used, where possible. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.c

RES: 2.05 Content </c>; rt="core.c"
```

Management objects MAY be discovered with the standard CoAP resource discovery. The implementation can add the encoded values of the object identifiers to /.well-known/core with rt="core.c.data". The available objects identified by the encoded values can be discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.data". Upon success, the return payload will contain the registered encoded values and their location. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.c.data

RES: 2.05 Content </c/BaAiN>; rt="core.c.data",
</c/CF_fA>; rt="core.c.data"
```

Lists of encoded values may become prohibitively long. It is discouraged to provide long lists of objects on discovery. Therefore, it is recommended that details about management objects are discovered by reading the YANG module information stored in for example the "ietf-comi-yang-library" module [I-D.veillette-core-cool-library]. The resource "/c/mod.uri" is used to retrieve the location of the YANG module library.

The module list can be stored locally on each server, or remotely on a different server. The latter is advised when the deployment of many servers are identical.

```
Local in example.com server:

REQ: GET example.com/c/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
  "mod.uri" : "example.com/c/modules"
}


Remote in example-remote-server:

REQ: GET example.com/c/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
  "moduri" : "example-remote-server.com/c/group17/modules"
}
```

Within the YANG module library all information about the module is stored such as: module identifier, identifier hierarchy, grouping, features and revision numbers.

9. Error Handling

In case a request is received which cannot be processed properly, the CoMI server MUST return an error message. This error message MUST contain a CoAP 4.xx or 5.xx response code, and SHOULD include additional information in the payload.

Such an error message payload is a text string, using the following structure:

```
CoMI error: xxxx "error text"
```

The characters xxxx represent one of the values from the table below, and the OPTIONAL "error text" field contains a human readable explanation of the error.

CoMI Error Code	CoAP Error Code	Description
0	4.xx	General error
1	4.13	Request too big
2	4.00	Response too big
3	4.00	Unknown identifier
4	4.00	Invalid value
5	4.05	Attempt to write read-only variable
6	5.01	No access
7	4.00	Wrong type
8	4.15	Unknown encoding
9	4.0	Wrong value
10	4.0	Not created
11	4.04	Resource unavailable
12	4.01	Authorization error
13	4.0	Bad attribute
14	4.0	Unknown attribute
15	4.0	Missing attribute

The CoMI error codes are motivated by the error-status values defined in [RFC3416], and the error tags defined in [I-D.ietf-netconf-restconf].

10. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. This requires integrity protection of both requests and responses, and depending on the application encryption.

CoMI re-uses the security mechanisms already available to CoAP as much as possible. This includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorization mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorization may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However some adaptations may still be required, to cater for CoMI's specific requirements.

11. IANA Considerations

'rt="core.c"' needs registration with IANA.

'rt="core.c.data"' needs registration with IANA.

'rt="core.c.moduri"' needs registration with IANA.

'rt="core.c.stream"' needs registration with IANA.

Content types to be registered:

- application/YANG-patch+cbor

- application/YANG-fetch+cbor

12. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification and specified CBOR encoding and use of hashes.

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR.

Timothy Carey has provided the text for Appendix B.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Juergen

Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

13. Changelog

Changes from version 00 to version 01

- o Focus on MIB only
- o Introduced CBOR, JSON, removed BER
- o defined mappings from SMI to xx
- o Introduced the concept of addressable table rows

Changes from version 01 to version 02

- o Focus on CBOR, used JSON for examples, removed XML and EXI
- o added uri-query attributes mod and con to specify modules and contexts
- o Definition of CBOR string conversion tables for data reduction
- o use of Block for multiple fragments
- o Error returns generalized
- o SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- o Added security considerations

Changes from version 03 to version 04

- o Added design considerations section
- o Extended comparison of management protocols in introduction
- o Added automatic generation of CBOR tables
- o Moved lowpan table to Appendix

Changes from version 04 to version 05

- o Merged SNMP access with RESTCONF access to management objects in small devices

- o Added CoMI architecture section
- o Added RESTCONf NETMOD description
- o Rewrote section 5 with YANG examples
- o Added server and payload size appendix
- o Removed Appendix C for now. It will be replaced with a YANG example.

Changes from version 04 to version 05

- o Extended examples with hash representation
- o Added keys query parameter text
- o Added select query parameter text
- o Better separation between specification and instance
- o Section on discovery updated
- o Text on rehashing introduced
- o Elaborated SMI MIB example
- o YANG library use described
- o use of BigEndian/LittleEndian in Hash generation specified

Changes from version 05 to version 06

- o Hash values in payload as hexadecimal and in URL in base64 numbers
- o Streamlined CoMI architecture text
- o Added select query parameter text
- o Data editing optional
- o Text on Notify added
- o Text on rehashing improved with example

Changes from version 06 to version 07

- o reduced payload size by removing JSON hierarchy

- o changed rehash handling to support small clients
- o added LWM2M comparison
- o Notification handling as specified in YANG
- o Added Patch function
- o Rehashing completely reviewed
- o Discover type of YANG name encoding
- o Added new resource types
- o Read-only servers introduced
- o Multiple updates explained

Changes from version 07 to version 08

- o Changed YANG Hash algorithm to use module name instead of prefix
- o Added rehash bit to allow return values to identify rehashed nodes in the response
- o Removed /c/mod.set resource since this is not needed
- o Clarified that YANG Hash is done even for unimplemented objects
- o YANG lists transported as CBOR maps of maps
- o Adapted examples with more CBOR explanation
- o Added CBOR code examples in new appendix
- o Possibility to use other than default stream
- o Added text and examples for Patch payload
- o Repaired some examples
- o Added appendices on hash clash probability and hash clash storage overhead

Changes from version 08 to version 09

- o Removed hash and YANG to CBOR sections

- o removed hashes from examples.
- o Added RPC
- o Added content query parameter.
- o Added default handling.
- o Listed differences with RESTCONF

Changes from version 09 to version 10. This is the merge of cool-01 with comi-09.

- o Merged with CoOL SIDs
- o Introduced iPATCH, PATCH and FETCH
- o Update of LWM2M comparison
- o Added appendix with module examples
- o Removed introductory text
- o Removed references

Changes from version 10 to version 11

- o Introduction streamlined
- o Error codes streamlined
- o Examples updated to latest SID values
- o Update of the YANG specifications in the appendix

14. References

14.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <http://www.rfc-editor.org/info/rfc5277>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <http://www.rfc-editor.org/info/rfc6243>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <http://www.rfc-editor.org/info/rfc7959>.

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

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.

[I-D.ietf-core-etch]
Stok, P., Bormann, C., and A. Sehgal, "Patch and Fetch Methods for Constrained Application Protocol (CoAP)", draft-ietf-core-etch-04 (work in progress), November 2016.

[I-D.bormann-appsawg-cbor-merge-patch]
Bormann, C. and P. Stok, "CBOR Merge Patch", draft-bormann-appsawg-cbor-merge-patch-00 (work in progress), March 2016.

[I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-03 (work in progress), October 2016.

14.2. Informative References

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <http://www.rfc-editor.org/info/rfc2578>.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <http://www.rfc-editor.org/info/rfc3410>.

[RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002, <http://www.rfc-editor.org/info/rfc3416>.

[RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <http://www.rfc-editor.org/info/rfc4293>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <http://www.rfc-editor.org/info/rfc6643>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., Koster, M., and C. Groves, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-07 (work in progress), December 2016.

[I-D.ietf-core-sid]
Somaraju, A., Veillette, M., Pelov, A., Turner, R., and A. Minaburo, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-00 (work in progress), October 2016.

[I-D.veillette-core-cool]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "Constrained Objects Language", draft-veillette-core-cool-02 (work in progress), July 2016.

[I-D.veillette-core-cool-library]
Veillette, M., "Constrained YANG Module Library", draft-veillette-core-cool-library-00 (work in progress), August 2016.

[XML] "Extensible Markup Language (XML)", Web http://www.w3.org/xml.

[OMA] "OMA-TS-LightweightM2M-V1_0-20131210-C", Web http://technical.openmobilealliance.org/Technical/current_releases.aspx.

[OMNA] "Open Mobile Naming Authority (OMNA)", Web http://http://technical.openmobilealliance.org/Technical/technical-information/omna.

[netconfcentral]
"NETCONF Central: library of YANG modules", Web http://www.netconfcentral.org/modulelist.

[mibreg] "Structure of Management Information (SMI) Numbers (MIB Module Registrations)", Web http://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml/.

[yang-cbor]
"yang-cbor Registry", Web https://github.com/core-wg/yang-cbor/tree/master/registry/.

Appendix A. YANG example specifications

This appendix shows 5 YANG example specifications taken over from as many existing YANG modules. The YANG modules are available from [netconfcentral]. Each YANG item identifier is accompanied by its SID shown after the "//" comment sign, taken from [yang-cbor].

A.1. ietf-system

Excerpt of the YANG module ietf-system [RFC7317].

```
module ietf-system {
  container system {                    // SID 1715
    container clock {                   // SID 1734
      choice timezone {
        case timezone-name {
          leaf timezone-name {          // SID 1735
            type timezone-name;
          }
        }
        case timezone-utc-offset {
          leaf timezone-utc-offset {    // SID 1736
            type int16 {
            }
          }
        }
      }
    }
    container ntp {                     // SID 1750
      leaf enabled {                    // SID 1751
        type boolean;
        default true;
      }
      list server {                     // SID 1752
        key name;
        leaf name {                     // SID 1755
          type string;
        }
        choice transport {
          case udp {
            container udp {             // SID 1757
              leaf address {            // SID 1758
                type inet:host;
              }
              leaf port {               // SID 1759
                type inet:port-number;
              }
            }
```

```
            }
          }
          leaf association-type {        // SID 1753
            type enumeration {
              enum server {
              }
              enum peer {
              }
              enum pool {
              }
            }
          }
          leaf iburst {                  // SID 1754
            type boolean;
          }
          leaf prefer {                  // SID 1756
            type boolean;
            default false;
          }
        }
      }
    container system-state {             // SID 1716
      container clock {                  // SID 1717
        leaf current-datetime {          // SID 1719
          type yang:date-and-time;
        }
        leaf boot-datetime {             // SID 1718
          type yang:date-and-time;
        }
      }
    }
  }
```

A.2. server list

Taken over from [RFC7950] section 7.15.3.

```
module example-server-farm {
  yang-version 1.1;
  namespace "urn:example:server-farm";
  prefix "sfarm";

  import ietf-yang-types {
    prefix "yang";
  }

  list server {                        // SID 60000
    key name;
    leaf name {                        // SID 60001
      type string;
    }
    action reset {                     // SID 60002
      input {
        leaf reset-at {                // SID 60003
          type yang:date-and-time;
          mandatory true;
        }
      }
      output {
        leaf reset-finished-at {       // SID 60004
          type yang:date-and-time;
          mandatory true;
        }
      }
    }
  }
}
```

A.3. interfaces

Excerpt of the YANG module ietf-interfaces [RFC7223].

```
module ietf-interfaces {
  container interfaces {                // SID 1505
    list interface {                    // SID 1533
      key "name";
      leaf name {                       // SID 1537
        type string;
      }
      leaf description {                // SID 1534
        type string;
      }
      leaf type {                       // SID 1538
        type identityref {
          base interface-type;
        }
        mandatory true;
      }

      leaf enabled {                    // SID 1535
        type boolean;
        default "true";
      }

      leaf link-up-down-trap-enable {   // SID 1536
        if-feature if-mib;
        type enumeration {
          enum enabled {
            value 1;
          }
          enum disabled {
            value 2;
          }
        }
      }
    }
  }
}
```

A.4. Example-port

Notification example defined within this document.

```
   module example-port {
           ...
           notification example-port-fault {   // SID 60010
             description
               "Event generated if a hardware fault on a
                line card port is detected";
             leaf port-name {                  // SID 60011
               type string;
               description "Port name";
             }
             leaf port-fault {                 // SID 60012
               type string;
               description "Error condition detected";
             }
           }
         }
```

A.5. IP-MIB

The YANG translation of the SMI specifying the IP-MIB [RFC4293], extended with example SID numbers, yields:

```
module IP-MIB {
  import IF-MIB {
    prefix if-mib;
  }
  import INET-ADDRESS-MIB {
    prefix inet-address;
  }
  import SNMPv2-TC {
    prefix smiv2;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import yang-smi {
    prefix smi;
  }
  import ietf-yang-types {
    prefix yang;
  }

  container ip {                            // SID 60020
    list ipNetToPhysicalEntry {             // SID 60021
      key "ipNetToPhysicalIfIndex ipNetToPhysicalNetAddressType ipNetToPhysicalN
etAddress";
      leaf ipNetToPhysicalIfIndex {         // SID 60022
        type if-mib:InterfaceIndex;
```

```
      }
      leaf ipNetToPhysicalNetAddressType {  // SID 60023
        type inet-address:InetAddressType;
      }
      leaf ipNetToPhysicalNetAddress {      // SID 60024
        type inet-address:InetAddress;
      }
      leaf ipNetToPhysicalPhysAddress {     // SID 60025
        type yang:phys-address {
          length "0..65535";
        }
      }
      leaf ipNetToPhysicalLastUpdated {     // SID 60026
        type yang:timestamp;
      }
      leaf ipNetToPhysicalType {            // SID 60027
        type enumeration {
          enum "other" {
            value 1;
          }
          enum "invalid" {
            value 2;
          }
          enum "dynamic" {
            value 3;
          }
          enum "static" {
            value 4;
          }
          enum "local" {
            value 5;
          }
        }
      }
      leaf ipNetToPhysicalState {           // SID 60028
        type enumeration {
          enum "reachable" {
            value 1;
          }
          enum "stale" {
            value 2;
          }
          enum "delay" {
            value 3;
          }
          enum "probe" {
            value 4;
          }
```

```
          enum "invalid" {
            value 5;
          }
          enum "unknown" {
            value 6;
          }
          enum "incomplete" {
            value 7;
          }
        }
      }
      leaf ipNetToPhysicalRowStatus {       // SID 60029
        type smiv2:RowStatus;
    }  // list ipNetToPhysicalEntry
  }  // container ip
}  // module IP-MIB
```

Appendix B. Comparison with LWM2M

B.1. Introduction

CoMI and LWM2M [OMA], both, provide RESTful device management services over CoAP. Differences between the designs are highlighted in this section.

The intent of the LWM2M protocol is to provide a single protocol to control and manage IoT devices. This means the IoT device implements and uses the same LWM2M agent function for the actuation and sensing features of the IoT device as well as for the management of the IoT device. The intent of CoMI Interface as described in the Abstract section of this document is to provide management of constrained devices and devices in constrained networks using RESTCONF and YANG. This implies that the device, although reusing the CoAP protocol, would need a separate CoAP based agent in the future to control the actuation and sensing features of the device and another CoMI agent that performs the management functions.

It should be noted that the mapping of a LWM2M server to YANG is specified in [YANGlwm2m]. The converted server can be invoked with CoMI as specified in this document.

For the purposes of managing IoT devices the following points related to the protocols compare how management resources are defined, identified, encoded and updated.

B.2. Defining Management Resources

Management resources in LWM2M (LWM2M objects) are defined using a standardized number. When a new management resource is defined, either by a standards organization or a private enterprise, the management resource is registered with the Open Mobile Naming Authority [OMNA] in order to ensure different resource definitions do not use the same identifier. CoMI, by virtue of using YANG as its data modeling language, allows enterprises and standards organizations to define new management resources (YANG nodes) within YANG modules without having to register each individual management resource. Instead YANG modules are scoped within a registered name space. As such, the CoMI approach provides additional flexibility in defining management resources. Likewise, since CoMI utilizes YANG, existing YANG modules can be reused. The flexibility and reuse capabilities afforded to CoMI can be useful in management of devices like routers and switches in constrained networks. However for management of IoT devices, the usefulness of this flexibility and applicability of reuse of existing YANG modules may not be warranted. The reason is that IoT devices typically do not require complex sets of configuration or monitoring operations required by devices like a router or a switch. To date, OMA has defined approximately 15 management resources for constrained and non-constrained mobile or fixed IoT devices while other 3rd Party SDOs have defined another 10 management resources for their use in non-constrained IoT devices. Likewise, the Constrained Object Language [I-D.veillette-core-cool] which is used by CoMI when managing constrained IoT devices uses YANG schema item identifiers, which are registered with IANA, in order to define management resources that are encoded using CBOR when targeting constrained IoT Devices.

B.3. Identifying Management Resources

As LWM2M and CoMI can similarly be used to manage IoT devices, comparison of the CoAP URIs used to identify resources is relevant as the size of the resource URI becomes applicable for IoT devices in constrained networks. LWM2M uses a flat identifier structure to identify management resources and are identified using the LWM2M object’s identifier, instance identifier and optionally resource identifier (for access to and object’s attributes). For example, identifier of a device object (object id = 3) would be "/3/0" and identification of the device object’s manufacturer attribute would be "/3/0/0". Effectively LWM2M identifiers for management resources are between 4 and 10 bytes in length.

CoMI is expected to be used to manage constrained IoT devices. CoMI utilizes the YANG schema item identifier[SID] that identify the resources. CoMI recommends that IoT device expose resources to

identify the data stores and event streams of the CoMI agent. Individual resources (e.g., device object) are not directly identified but are encoded within the payload. As such the identifier of the CoMI resource is smaller (4 to 7 bytes) but the overall payload size isn't smaller as resource identifiers are encoded on the payload.

B.4. Encoding of Management Resources

LWM2M provides a separation of the definition of the management resources from how the payloads are encoded. As of the writing of this document LWM2M encodes LWM2M encodes payload data in Type-length-value (TLV), JSON or plain text formats. JSON encoding is the most common encoding scheme with TLV encoding used on the simplest IoT devices. CoMI's use of CBOR provides a more efficient transfer mechanism [RFC7049] than the current LWM2M encoding formats.

In situations where resources need to be modified, CoMI uses the CoAP PATCH operation resources only require a partial update. LWM2M does not currently use the CoAP PATCH operation but instead uses the CoAP PUT and POST operations which are less efficient.

Authors' Addresses

Peter van der Stok
consultant

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

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

Email: andy@yumaworks.com

Michel Veillette
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

core
Internet-Draft
Intended status: Informational
Expires: September 16, 2016

P. van der Stok
A. Sehgal
Consultant

March 15, 2016

Patch Method for Constrained Application Protocol (CoAP)
draft-vanderstok-core-patch-03

Abstract

The existing Constrained Application Protocol (CoAP) PUT method only allows a complete replacement of a resource. This does not permit applications to perform partial resource modifications. In case of resources with larger or complex data, or in situations where a resource continuity is required, replacing a resource is not an option. Several applications using CoAP will need to perform partial resource modifications. This proposal adds new CoAP methods, PATCH and iPATCH, to modify an existing CoAP resource partially.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This specification defines the new Constrained Application Protocol (CoAP) [RFC7252] methods, PATCH and iPATCH, which are used to apply partial modifications to a resource.

PATCH is also specified for HTTP in [RFC5789]. Most of the motivation for PATCH described in [RFC5789] also applies here. iPATCH is the idem-potent version of PATCH.

The PUT method exists to overwrite a resource with completely new contents, and cannot be used to perform partial changes. When using PUT for partial changes, proxies and caches, and even clients and servers, may get confused as to the result of the operation. PATCH was not adopted in an early design stage of CoAP, however, it has become necessary with the arrival of applications that require partial updates to resources (e.g. [I-D.vanderstok-core-comi]). Using PATCH avoids transferring all data associated with a resource in case of modifications, thereby not burdening the constrained communication medium.

This document relies on knowledge of the PATCH specification for HTTP [RFC5789]. This document provides extracts from [RFC5789] to make independent reading possible.

1.1. Requirements Language

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

1.2. Terminology and Acronyms

This document uses terminology defined in [RFC5789] and [RFC7252].

2. PATCH (iPATCH) Method

The PATCH (iPATCH) method requests that a set of changes described in the request payload is applied to the target resource of the request. The set of changes is represented in a format identified by a media type. If the Request-URI does not point to an existing resource, the server MAY create a new resource with that URI, depending on the patch document type (whether it can logically modify a null resource) and permissions, etc. Creation of a new resource would result in a 2.01 (Created) Response Code dependent of the patch document type.

Restrictions to a PATCH (iPATCH) can be made by including the If-Match or If-None-Match options in the request (see Section 5.10.8.1 and 5.10.8.2 of [RFC7252]). If the resource could not be created or modified, then an appropriate Error Response Code SHOULD be sent.

The difference between the PUT and PATCH requests is extensively documented in [RFC5789].

PATCH is not safe and not idempotent conformant to HTTP PATCH specified in [RFC5789].

iPATCH is not safe but idempotent conformant to CoAP PUT specified in [RFC7252], Section 5.8.3.

An iPATCH request is idempotent to prevent bad outcomes from collisions between two iPATCH requests on the same resource in a similar time frame. These collisions can be detected with the MessageId and the source end-point provided by the CoAP protocol (see section 4.5 of [RFC7252].

PATCH and iPATCH are both atomic. The server MUST apply the entire set of changes atomically and never provide a partially modified representation to a concurrently executed GET request. Given the constrained nature of the servers, most servers will only execute CoAP requests consecutively, thus preventing a concurrent partial overlapping of request modifications. Resuming, modifications MUST NOT be applied to the server state when an error occurs or only a

partial execution is possible on the resources present in the server. When the PATCH request is over-specified (i.e. Request specifies modifications to attributes which do not exist in the server), The server MAY execute all modifications to existing attributes and return a response code 2.02 Accepted.

The atomicity applies to a single server. When a PATCH (iPATCH) request is multicast to a set of servers, each server can either execute all required modifications or not. It is not required that all servers execute all modifications or none. An Atomic Commit protocol that provides multiple server atomicity, is out of scope.

A PATCH (iPATCH) response can invalidate a cache conformant with the PUT response. Caching behaviour as function of the valid 2.xx response codes for PATCH (iPATCH) are:

A 2.01 (Created) response invalidates any cache entry for the resource indicated by the Location-* Options; the payload is a representation of the action result.

A 2.04 (Changed) response invalidates any cache entry for the target resource; the payload is a representation of the action result.

There is no guarantee that a resource can be modified with PATCH (iPATCH). Servers are required to support a subset of the content formats as specified in sections 12.3 and 5.10.3 of [RFC7252]. Servers MUST ensure that a received PATCH payload is appropriate for the type of resource identified by the target resource of the request.

Clients MUST choose to use PATCH (iPATCH) rather than PUT when the request affects partial updates of a given resource.

PATCH (iPATCH) MUST not be used to restore default values to resource attributes which are not specified in the payload. PATCH (iPATCH) specifically guarantees that unspecified resource attributes are not changed.

2.1. A Simple PATCH (iPATCH) Example

The example is taken over from [RFC6902], which specifies a JSON notation for PATCH operations. A resource located at www.example.com/object contains a target JSON document.

```
JSON document original state
            {
              "x-coord": 256,
              "y-coord": 45",
              "foo": ["bar","baz"]
            }

 REQ:
     iPATCH CoAP://www.example.com/object
Content-Format: application/json-patch+json
         [
             { "op":"replace","path":"x-coord","value":45}
         ]
RET:
     CoAP 2.04 Changed

JSON document final state
            {
               "x-coord": 45,
               "y-coord": 45,
               "foo": ["bar","baz"]
             }
```

This example illustrates use of an idempotent modification to the x-coord attribute of the existing resource "object". The 2.04 (Changed) response code is conform with the CoAP PUT method.

The same example using the Content-Format application/merge-patch+json from [RFC7396] looks like:

```
JSON document original state
            {
              "x-coord": 256,
              "y-coord": 45,
              "foo": ["bar","baz"]
            }

 REQ:
     iPATCH CoAP://www.example.com/object
Content-Format: application/merge-patch+json
     { "x-coord":45}
RET:
     CoAP 2.04 Changed

JSON document final state
            {
               "x-coord": 45,
               "y-coord": 45,
               "foo": ["bar","baz"]
             }
```

The examples show the use of the iPATCH method, but the use of the PATCH method must have led to the same result. Below a non-idempotent modification is shown. Because the action is non-idempotent, iPATCH returns an error, while PATCH executes the action.

```
JSON document original state
            {
              "x-coord": 256,
              "y-coord": 45",
              "foo": ["bar","baz"]
            }

 REQ:
     iPATCH CoAP://www.example.com/object
Content-Format: application/json-patch+json
         [
             { "op":"add","path":"foo/1","value":"bar"}
         ]
RET:
     CoAP 4.12 Precondition Failed

JSON document final state is unchanged

REQ:
     PATCH CoAP://www.example.com/object
Content-Format: application/json-patch+json
         [
             { "op":"add","path":"foo/1","value":"bar"}
         ]
RET:
     CoAP 2.04 Changed

JSON document final state

            {
               "x-coord": 45,
               "y-coord": 45,
               "foo": ["bar","bar","baz"]
             }
```

2.2. Response Codes

PATCH (iPATCH) for CoAP adopt the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252].

2.3. Option Numbers

PATCH for CoAP adopts the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

3. Error Handling

A PATCH (iPATCH) request may fail under certain known conditions. These situations should be dealt with as expressed below.

Malformed PATCH (iPATCH) payload: If a server determines that the payload provided with a PATCH (iPATCH) request is not properly formatted, it can return a 4.00 (Bad Request) CoAP error. The definition of a malformed payload depends upon the CoAP Content-Format specified with the request.

Unsupported PATCH (iPATCH) payload: In case a client sends payload that is inappropriate for the resource identified by the Request-URI, the server can return a 4.15 (Unsupported Content-Format) CoAP error. The server can determine if the payload is supported by checking the CoAP Content-Format specified with the request.

Unprocessable request: This situation occurs when the payload of a PATCH request is determined as valid, i.e. well-formed and supported, however, the server is unable to or incapable of processing the request. The server can return a 4.22 (Unprocessable Entity) CoAP error. More specific scenarios might include situations when:

* the server has insufficient computing resources to complete the request successfully -- 4.13 (Request Entity Too Large) CoAP Response Code,

* the resource specified in the request becomes invalid by applying the payload -- 4.06 (Not Acceptable) CoAP Response Code,

In case there are more specific errors that provide more insight into the problem, then those should be used.

Resource not found: The 4.04 (Not Found) error should be returned in case the payload of a PATCH request cannot be applied to a non-existent resource.

Failed precondition: In case the client uses the conditional If-Match or If-None-Match option to define a precondition for the PATCH request, and that precondition fails, then the server can return the 4.12 (Precondition Failed) CoAP error.

Request too large: If the payload of the PATCH request is larger than a CoAP server can process, then it can return the 4.13 (Request Entity Too Large) CoAP error.

Conflicting state: If the modification specified by a PATCH (iPATCH) request causes the resource to enter an inconsistent state that the server cannot resolve, the server can return the 4.09 (Conflict) CoAP response. The server SHOULD generate a payload that includes enough information for a user to recognize the source of the conflict. The server MAY return the actual resource state to provide the client with the means to create a new consistent resource state. Such a situation might be encountered when a structural modification is applied to a configuration data-store, but the structures being modified do not exist.

Concurrent modification: Resource constrained devices might need to process requests in the order they are received. In case requests are received concurrently to modify the same resource but they cannot be queued, the server can return a 5.03 (Service unavailable) CoAP response code.

Conflict handling failure: If the modification implies the reservation of resources or the waiting on conditions to become true, leading to a too long request execution time, the server can return 5.03 (service unavailable) response code.

It is possible that other error situations, not mentioned here, are encountered by a CoAP server while processing the PATCH request. In these situations other appropriate CoAP status codes can also be returned.

4. Security Considerations

This section analyses the possible threats to the CoAP PATCH (iPATCH) protocol. It is meant to inform protocol and application developers about the security limitations of CoAP PATCH (iPATCH) as described in this document. The security consideration of section 15 of [RFC2616], section 11 of [RFC7252], and section 5 of [RFC5789] also apply.

The security considerations for PATCH (iPATCH) are nearly identical to the security considerations for PUT ([RFC7252]). The mechanisms used for PUT can be used for PATCH (iPATCH) as well.

PATCH (iPATCH) is secured following the CoAP recommendations as specified in section 9 of [RFC7252]. When more appropriate security techniques are standardized for CoAP, PATCH (iPATCH) can also be secured by those new techniques.

5. IANA Considerations

The entry with name PATCH in the sub-registry, "CoAP Method Codes", is 0.05. The entry with name iPATCH in the sub-registry, "CoAP Method Codes", is 0.06. The additions will follow the "IETF Review or IESG Approval" procedure as described in [RFC5226].

A new response code must be entered to the sub-registry "CoAP response codes" which apply to the methods PATCH and iPATCH.

Code 4.09 with Description "Conflict" and described in this specification.

The addition to this sub-registry will follow the "IETF Review or IESG Approval" as described in [RFC5226].

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are needed for the following media type formats: "application/json-patch+json" [RFC6902], and "application/merge-patch+json" [RFC7396].

6. Acknowledgements

Klaus Hartke has pointed out some essential differences between CoAP and HTTP. We are grateful for discussions with Christian Amsuss, Carsten Bormann, Timothy Carey, Paul Duffy, Kovatsch Matthias, Michel Veillette, Michael Verschoor, Thomas Watteyne, and Gengyu Wei.

7. Change log

When published as a RFC, this section needs to be removed.

Version 0 to version 1:

- o Changed patch motivation text.
- o Removed sub-resource concept.
- o Updated cache handling.
- o Extended example.
- o Update of error handling.

Version 1 to version 2

- o section 3 rephrased use of error 4.09

- o added conflict handling failure
- o added idempotent iPATCH method

Version 2 to version 3

- o added line about default values
- o content-Type to content_Format
- o extended Patch example with non-idempotent request

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <http://www.rfc-editor.org/info/rfc5789>.

[RFC6902] Bryan, P., Ed. and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902, April 2013, <http://www.rfc-editor.org/info/rfc6902>.

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

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

8.2. Informative References

[I-D.vanderstok-core-comi]
Stok, P. and A. Bierman, "CoAP Management Interface", draft-vanderstok-core-comi-09 (work in progress), March 2016.

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Anuj Sehgal
Consultant

Email: anuj@iurs.org

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: March 31, 2016

T. Zotti
Philips Research
P. van der Stok
Consultant
E. Dijk
Philips Research

September 28, 2015

Sleepy CoAP Nodes
draft-zotti-core-sleepy-nodes-04

Abstract

Control networks rely on application protocols like CoAP to enable RESTful communications in constrained environments. Many of these networks make use of "Sleepy Nodes": battery powered devices that switch off their (radio) interface during most of the time to conserve battery energy. As a result of this, Sleepy Nodes cannot be reached most of the time. This fact prevents using normal communication patterns as specified in the CoRE group, since the server-model is not applicable to these devices. This document discusses and specifies an architecture to support Sleepy Nodes such as battery-powered sensors in mesh networks with the goal of proposing a standardisation solution for Sleepy Node proxies.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Control networks rely on application protocols such as CoAP to enable RESTful communications in constrained environments. Many of these networks feature "Sleepy Nodes": battery-powered nodes which switch on/off their communication interface to conserve battery energy. As a result of this, Sleepy Nodes cannot be reached most of the time. This fact prevents using normal communication patterns as specified by the CoRE group, since the server model is clearly not applicable to the most energy constrained devices.

This document discusses and specifies an architecture to support Sleepy Nodes such as battery-powered sensors in wireless networks. The proposed solution makes use of a Proxy Node to which a Sleepy Node delegates part of its communication tasks while it is not accessible in the wireless network. Direct interactions between Sleepy Nodes and non-Sleepy Nodes are only possible, when the Sleepy Node initiates the communication.

Earlier related documents treating the Sleepy Node subject are the CoRE mirror server [I-D.vial-core-mirror-server] and the Publish-Subscribe in the Constrained Application Protocol (CoAP) [I-D.koster-core-coap-pubsub]. Both documents describe the interfaces to the proxy accompanying the Sleepy Node. Both make use of the observe option discussed in [I-D.ietf-core-observe]. This document describes the roles of the nodes communicating with the Sleepy Node and/or its proxy. The draft describes the differences between the concepts supporting the Sleepy Node, and the concepts underlying the PubSub paradigm.

The draft relies heavily on the concepts introduced by the Resource Directory [I-D.ietf-core-resource-directory], and describes how the Sleepy Node profits of the introduction of a Resource Directory into the network.

The issues that need to be addressed to provide support for Sleepy Nodes in Control networks are summarized in Section 1.1. Section 2 provides a set of use case descriptions that introduce communication patterns to be used in home and building control scenarios. Section 4, Section 5,Section 6, and Section 7 specify interfaces to support each of these scenarios. Many interface specifications and examples are taken over from [I-D.vial-core-mirror-server].

1.1. Problem statement

During typical operation, a Sleepy Node has its radio disabled and the CPU may be in a sleeping state. If an external event occurs (e.g. person walks into the room activating a presence sensor), the

CPU and radio are powered back on and they send out a message to another node, or to a group of nodes. After sending this message, the radio and CPU are powered off again, and the Sleepy Node sleeps until the next external event or until a predefined time period has passed. The main problems when introducing Sleepy Nodes into a wireless network are as follows:

Problem 1: How to contact a Sleepy Node that has its radio turned off most of the time for:

- Writing configuration settings.

- Reading out sensor data, settings or log data.

- Configuring additional event destination nodes or node groups.

Problem 2: How to discover a Sleepy Node and its services, while the node is asleep:

- Direct node discovery (CoAP GET /.well-known/core as defined in [RFC7252]) does not find the node with high probability.

- Mechanisms may be needed to provide, as the result of node discovery, the IP address of a Proxy instead of the IP address of the node directly.

Problem 3: How a Sleepy Node can convey data to a node or groups of nodes, with good reliability and minimal energy consumption.

1.2. Assumptions

The solution architecture specified here assumes that a Sleepy Node has enough energy to perform bidirectional communication during its normal operational state. This solution may be applicable also to extreme low-power devices such as solar powered sensors as long as they have enough energy to perform commissioning and the initial registration steps. These installation operations may require, in some cases, an additional source of power. Since a Sleepy Node is unreachable for relatively long periods of times, the data exchanges in the interaction model are always initiated by a Sleepy Node when its sleep period ends.

1.3. Requirements Language

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

This document assumes readers are familiar with the terms and concepts discussed in [RFC7252],[RFC5988], [I-D.ietf-core-resource-directory], [I-D.ietf-core-interfaces],[I-D.ietf-core-observe] and [I-D.vial-core-mirror-server].

In addition, this document makes use of the following additional terminology:

Sleepy Node: a battery-powered node which does the on/off switching of its communication interface with the purpose of conserving battery energy

Sleeping/Asleep: A Sleepy Node being in a "sleeping state" i.e. its network interface is switched off and a Sleepy Node is not able to send or receive messages.

Awake/Not Sleeping: A Sleepy Node being in an "awake state" i.e. its network interface is switched on and the Sleepy Node is able to send or receive messages.

Wake up reporting duration: the duration between a wake up from a Sleepy Node and the next wake up and report of the same Node.

Proxy: any node that is configured to, or selected to, perform communication tasks on behalf of one or more Sleepy Nodes.

Regular Node: any node in the network which is not a Proxy or a Sleepy Node.

1.4. Acronyms

This Internet-Draft contains the following acronyms:

DTLS: Datagram Transport Layer Security

EP: Endpoint

MC: Multicast

RD: Resource Directory

2. Use cases and architecture

To describe the application viewpoint of the solution, we introduce some example scenarios for the various interactions shown in Figure 1. The figure assigns the following roles taken up by a regular node:

- o Reading Node: any regular node that reads information from the Sleepy Node.

- o Configuring Node: any regular node that writes information/ configuration into Sleepy Node(s). Examples of configuration are new thresholds for a sensor or a new value for the wake-up cycle time.

- o Discovering Node: any regular node that performs discovery of the nodes in a network, including Sleepy Nodes.

- o Destination Node: any regular node or node in a group that receives a message that is generated by the Sleepy Node.

- o Server Node: an optional server that the Sleepy Node knows about, or is told about, which is used to fetch information/configuration/firmware updates/etc.

- o Discovery Server: an optional server that enables nodes to discover all the devices in the network, including Sleepy Nodes, and query their capabilities. For example, a Resource Directory server as defined in [I-D.ietf-core-resource-directory] or a DNS-SD server as defined in [RFC6763]. For the rest of this document the discovery server is a Resource Directory. Specifically, the functionalities of the Resource Directory related to the architecture presented in this Internet-Draft are described in more details in Section 4.

- o Delegated resource is the copy at the Proxy of a resource present in the Sleepy Node.

2.1. Node interactions and use cases

```
                 +------------+             +-------------+
                 | Discovery  | <-DISCOVERY-| Discovering |
                 | server     |             |    Node     |
                 | (Optional) |             +-------------+
                 +------------+                      |
                                                     |
                                       .--DISCOVERY--'  +---------+
                                       |                | Reading |
                                       |            .---|   Node  |
                                       v            |   +---------+
   +---------+               +-----------+         |
   | Sleepy  |---REPORT(A)-->|           |<--READ--' +-------------+
   |  Node   |---READ------->|   Proxy   |<--WRITE---| Configuring |
   |         |---WRITE------>|           |           |    Node     |
   +---------+               +-----------+           +-------------+
     |     |                           |             +-------------+
     |     |                           '---REPORT(B)->| Destination |
     |     '-----DIRECT REPORT---------------------->|     Node    |
     |                                               +-------------+
     |                                          +-----------+
     '------------READ--------------------------->|  Server   |
                                                  |   Node    |
                                                  +-----------+
```

Figure 1: Interaction model for Sleepy Nodes in IP-based networks

The interactions visualized in Figure 1 are discussed and motivated with their use cases. The arrows in the figure indicate that the initiative for an interaction is taken by the source of the arrow.

DISCOVERY Interaction: a Discovering Node discovers Sleepy Node(s)via Proxy or Discovery Server; for example:

- A Discovering Node wants to discover given services related to a group of deployed sensors by sending a multicast to /.well-known/core. It gets responses for the sleeping sensors from the Proxy nodes.

- During commissioning phase, a discovering node queries a Discovery Server to find all the proxies providing a given service.

REPORT Interaction: On request of a Destination Node or because of configuration settings which have instructed the Node to do so, a Node sends a sequence of event notifications to destination Node(s), (A) directly or (B) via Proxy; for example:

- A battery-powered sensor sends a notification with "battery low" event directly to a designated Destination Node (REPORT(A)).

- A battery-powered occupancy sensor detects an event "people present", switches on the radio and multicasts an "ON" command to a group of lights (REPORT(A)).

- A battery-powered temperature sensor reports periodically the room temperature to a proxy Node (REPORT(A)). The proxy node reports to all associated HVAC destination nodes when the temperature change deviates from a predefined range (REPORT(B)).

WRITE Interaction: A node sends a request to a proxy to set a value.

o A Sleepy Node WRITES to the proxy; for example:

  - A battery-powered sensor wants to extend the registration lifetime of its delegated resource at the Proxy.

o A configuring Node WRITEs information to a Proxy; for example:

  - A configuring Node changes the reporting frequency of a deployed sensor by contacting the Proxy node to which the sensor is registered.

  - Sensor firmware is upgraded. A configuring Node pushes firmware data blocks to the Proxy, which pushes the blocks to the Sleepy Node.

  - A configuring Node adds a new subscription to an operational sensor via the Proxy. From that moment on, the new Node receives also the sensor events and status updates from the sensor.

READ Interaction: A node sends a read request to a node that returns a value.

o Sleepy Node sends a read request to a server Node; for example:

  - A sensor (periodically) updates internal data tables by fetching it from a predetermined remote node.

  - A sensor (periodically) checks for new firmware with a remote node. If new firmware is found, the sensor switches to a non-sleepy operation mode, and fetches the data.

o A Sleepy Node sends a read request to its proxy; for example:

- A sensor (periodically) checks with his Proxy availability of configuration updates or changes of its delegated resources (e.g. a sensor may detect in this way that a configuring Node has changed its name or modified its reporting frequency).

o A reading Node sends a read request to a proxy; for example:

- A Node (e.g. in the backend) requests the status of a deployed sensor, e.g. asking the sensor state and/or firmware version and/or battery status and/or its error log. The Proxy returns this information.

- A Node requests a Proxy when a Sleepy sensor was 'last active' (i.e. identified as being awake) in the network.

2.2. Architecture

The architecture associated with the support of Sleepy Nodes is illustrated in Figure 2. Three High level interfaces are shown.

```
       direct        synchronize      delegate
         |               |               |
+----+   |   +--------+  |   +-------+   |   +----+
| EP |---|---| sleepy |---|---| proxy |---|---| EP |
+----+   |   +--------+  |   +-------+   |   +----+
         |               |               |
```

Figure 2: Architecture of Sleepy Node support

o Direct interface: it allows the Sleepy Node to communicate directly to endpoints (i.e. for sending or reading information). The operations performed via this interface are always initiated by the Sleepy Node when its sleep period ends.

o Delegate interface: via this interface the Proxy exposes the values of delegated resources to interested endpoints on behalf of the Sleepy Node. The same interface is used by endpoints which want to communicate with the Sleepy Node (e.g. for reading or writing information).

o Synchronize interface: used by Sleepy Node and Proxy to synchronize values of delegated resources. Through this interface operations as discovery of the Proxy, registration, initialization and update of resources at the Proxy are performed, along with a de-registration operation to explicitly remove resources already registered to the Proxy.

The interfaces consist of a set of functions which together realize the interactions described in Section 2.1.

Endpoints and the proxy communicate with a Resource Directory (RD) to discover resources of the Sleepy Node and delegated resources on the proxy (not shown in the Figure 2).

2.3. Example contents

The examples presented in this specification make use of a smart temperature sensor the resources of which are defined below using Link Format [RFC6690]. Three resources are dedicated to the Device Description (manufacturer, model, name) and one contains the current temperature in degree Celsius.

```
</dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</dev/n>;rt="ipso.dev.n";if="core.p",
</sen/temp>;rt="ucum.Cel";if="core.s"
```

3. Design motivation

The Sleepy Node stack features a CoAP interface, to make the Sleepy Node part of the IP-based network. Adding CoAP with a transport protocol increases the possibilities to configure the Sleepy Node within the network. The increased energy consumption coming from the overhead of the CoAP and IP headers can be acceptable in many cases.

The proxy and Sleepy Node make use of the /.well-known/core resource to handle discovery during network initialization. Using the Resource Directory during operation of the Sleepy Node reduces its participation in the discovery traffic.

A Sleepy Node delegates its resources to a proxy. The proxy functionality extends the functionality of the RD, because the proxy handles the value of the resource, and the RD does not. A proxy may support multiple Sleepy Nodes. A Sleepy Node may also delegate its resources to multiple proxies. A node can select a proxy that handles the resource of the Sleepy Node of choice.

The complexity of the discovery and delegation interfaces is minimized by reusing the RD interface as much as possible.

4. Interactions involving Resource Directory

It is assumed that the Proxy has a resource type rt="core.sp", where sp stands for sleepy proxy.

In order to become fully operational in a network and to communicate over the functional interfaces shown in Figure 2, a Sleepy Node and the Proxy need to perform operations via the Registration interface of the RD:

- Discovery of Proxy via RD. The Sleepy Node MAY discover the Proxy by sending a request to the RD to return all EP with rt=core.sp.

- Register existence of Proxy. When a RD is present and a Sleepy Node has registered itself to a Proxy (see Section 5.2), the Proxy MUST register the Sleepy Node at the RD and MUST keep this registration up-to-date.

- Register delegated resources. When a RD is present, the Proxy MUST register the delegated resources at the RD and keep them up-to date.

A Configuring Endpoint (often part of a so-called Commissioning Tool) registers the services that are reported directly by the Sleepy Node in the resource directory, by registering the resource type and the multicast address. The multicast address can be associated with a group as described in [I-D.ietf-core-resource-directory].

A discovering Endpoint can discover one or more Sleepy Node resources via the Resource Directory.

```
+-------------+                          +-----------------+
| Configuring |                          | Discovering     |---.
| Endpoint    |                          | Endpoint        |   |
+-------------+                          +-----------------+   |
      |                                                         |
      |                    +------------+                       |
      .-Register MC------>|            |<--Discover resources -.
                          | Resource   |
                          | Directory  |<--Register Proxy -----.
      .-Proxy Discovery-->|            |<--Register resources -.
      |                   +------------+                       |
      |                                                         |
+---------+                               +-----------+         |
| Sleepy  |                               |   Proxy   |---------'
| Node    |                               |           |
+---------+                               +-----------+
```

Figure 3: Interactions involving Resource Directory

5. Synchronize interface

The functions of the synchronize interface implemented by the Proxy are described in this section.

5.1. Sleepy Node discovers proxy

A Sleepy Node can discover the proxy in two ways:

- via the CoAP interface [RFC7390] by sending a multicast message to discover an endpoint with rt=core.sp.

- via RD as already described in Section 4.

The following example shows a sleeping endpoint discovering a proxy using this interface, thus learning that the base Proxy resource, where the Sleepy Node resources are registered, is at /sp.

```
Sleepy                                            Proxy
  |                                                 |
  | ----- GET /.well-known/core?rt=core.sp ------>  |
  |                                                 |
  |                                                 |
  | <---- 2.05 Content "</sp>; rt="core.sp" ------  |
  |                                                 |
```

```
Req: GET coap://[ff02::1]/.well-known/core?rt=core.sp
Res: 2.05 Content
</sp>;rt="core.sp"
```

The use of /sp is recommended and not compulsory.

5.2. Registration at a Proxy

Once a Sleepy Node has discovered a Proxy by means of one of the procedures described in Section 5.1, the registration step can be performed. To perform registration, a Sleepy Node sends to the Proxy Node a CoAP POST request containing a description of the resources to be delegated to the Proxy as the message payload in the CoRE Link Format [RFC6690]. The description of the resource includes the Sleepy Node identifier, its domain and the lifetime of the registration.

Upon successful registration a Proxy creates a new delegated resource or updates an existing delegated resource and returns its location. The resources specified by the Sleepy Node during registration are created with path that has as prefix the base Proxy resource path (e.g. /sp). The registration interface MUST be implemented to be

idempotent, so that registering twice with the same endpoint parameter does not create multiple delegated resources. The delegated resource SHOULD implement the Interface Type CoRE Link List defined in [I-D.ietf-core-interfaces]. A GET request on this resource MUST return the list of delegated resources for the corresponding Sleepy Node.

After successful registration, a Proxy SHOULD enable resource discovery for the new resources by updating its "/.well-known/core" resource. A Proxy MUST wait for the initial representation of a resource before it can be visible during resource discovery. The top level delegated resource MUST be published in "/.well-known/core" to enable the discovery of the resources via RD as described in Section 4. Resources of a delegated container SHOULD be discoverable either directly in "/.well-known/core" or indirectly through gradual reveal from the delegated resource. The Web Link of a delegated resource MUST contain an "ep" attribute with the value of the End-Point parameter received during registration.

A Proxy MAY be configured to register the Sleepy Node’s resources in a RD. In this case, a Sleepy Node MUST NOT register the resources in a RD by itself since it is the responsibility of the Proxy to perform the registration in the RD on behalf of the Sleepy Node. Since each Sleepy Node may register resources with different lifetimes, a Proxy MUST register the resources of a given Sleepy Node in a dedicated path of the RD.

In case a Sleepy Node delegates its own resources to more than one Proxy and each Proxy registers the Sleepy Node’s resource in a RD, the RD entries from the different Proxies for the same Sleepy Node risk to overlap.

To avoid this problem, a Proxy MUST create its own resource path to register the resources of a Sleepy Node on the RD.

The new path name is typically formed by concatenating the Proxy’s endpoint identifier with the path in use. This precaution ensures that the ep identifier of a Sleepy Node is unique for each resource path in the RD.

Implementation note: It is not recommended to reuse the value of the ep parameter in the URI of the delegated resource. This parameter may be a relatively long identifier to guarantee global uniqueness (e.g. EUI64) and would generate inefficient URIs on the Proxy where only a local handler is necessary.

The following example shows a Sleepy Node registering with a Proxy.

```
Sleepy                                                    Proxy
  |                                                         |
  | --- POST /sp?ep=0224e8fffe925dcf;rt=sensor "</dev..."---> |
  |                                                         |
  |                                                         |
  | <-- 2.01 Created Location: /sp/0 ----------------------- |
  |                                                         |
```

```
Req: POST coap://sp.example.org/sp?ep=0224e8fffe925dcf;rt=sensor
Etag: 0x3f
Payload:
</dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</dev/n>;rt="ipso.dev.n";if="core.p",
</sen/temp>;rt="ucum.Cel";if="core.s"

Res: 2.01 Created
Location: /sp/0
```

The delegated resource has been created with path /sp/0 on the Proxy in the example above. The path to the ep can be discovered as shown below:

```
Req: GET coap://sp.example.org/.well-known/core
Res: 2.05 Content
</sp>;rt="core.sp",
</sp/0>;ep="0224e8fffe925dcf";rt="sensor"
```

A node can discover the delegated resources of the ep as shown below:

```
Req: GET coap://sp.example.org/sp/0
Res: 2.05 Content
Payload:
</sp/0/dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</sp/0/dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</sp/0/dev/n>;rt="ipso.dev.n";if="core.p",
</sp/0/sen/temp>;rt="ucum.Cel";if="core.s"
```

Once the resources are registered in the Proxy, the Proxy registers the delegated resources in the RD.

```
Proxy                                                        RD
  |                                                           |
  | --- POST /rd?ep=0224e8fffe925dcf "</sp/0..." -------->    |
  |                                                           |
  | <-- 2.01 Created Location: /rd/6534 -------------------   |
  |                                                           |
```

```
Req: POST coap://rd.example.org/rd?ep=0224e8fffe925dcf
Etag: 0x6a
Payload:
</sp/0/dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</sp/0/dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</sp/0/dev/n>;rt="ipso.dev.n";if="core.p",
</sp/0/sen/temp>;rt="ucum.Cel";if="core.s"

Res: 2.01 Created
Location: /rd/6534
```

5.3. De-registration at a Proxy

Sleepy Node resources in the Proxy are kept active for the period indicated by the lifetime parameter. The Sleepy Node is responsible for refreshing the delegated resource within this period using either the registration or update function (see Section 5.5 of the Synchronize interface). Once a delegated resource has expired, the Proxy deletes all resources associated to that resource and updates its "/.well-known/core" resource. When the Proxy resources are also registered in a RD, the RD and delegated resources are supposed to have the same lifetime. Consequently, when the delegated resource expires, a Proxy MAY let the RD resource expire too instead of explicitly deleting it. When the delegated resource is deleted by means of explicit de-registration operation then also the RD resource MUST be explicitly removed.

A Proxy could lose or delete the delegated resource associated to a Sleepy Node without sending an explicit notification (e.g. after reboot). A Sleepy Node SHOULD be able to detect this situation by processing the response code while using the Sleepy Node Operation or Update interface. Especially an error code "4.04 Not Found" SHOULD cause the Sleepy Node to register again. A Sleepy Node MAY also register with multiple proxies to alleviate the risk of interruption of service.

5.4. Initialization of delegated resource

Once registration has been successfully performed, the Sleepy Node must initialize the delegated resource. To send the initial contents (e.g. values, device name, manufacturer name) of the delegated resources to the Proxy, the Sleepy Node uses CoAP PUT repeatedly.

The basic interface is specified as follows:

Interaction: Sleepy -> Proxy

Method: PUT

URI Template: /{+location}{+resource}{?lt}

URI Template Variables:

location := This is the Location path returned by the Proxy as a result of a successful registration.

resource := This is the relative path to a delegated resource managed by the registered Sleepy Node.

lt := Lifetime (optional). The number of seconds by which the lifetime of the whole delegated resource is extended. Range of 1-4294967295. If no lifetime is included, the current remaining lifetime stays unchanged.

Request Content-Type: Defined at registration

Response Content-Type: Defined at registration for GET method. application/link-format for PUT method if at least one of the mutable resources has been updated since the last PUT request.

Etag: The Etag option MAY be included to allow clients to validate a resource on multiple Proxies.

Success: 2.01 "Created", the request MUST include the initial representation of the delegated resource.

Success: 2.04 "Changed", the request MUST include the new representation of the delegated resource.

Success: 2.05 "Content", the response MUST include the current representation of the delegated resource.

Failure: 4.00 "Bad Request". Malformed request.

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

The following example describes how a Sleepy Node can initialize the resource containing its manufacturer name just after registration.

```
Sleepy                                          Proxy
  |                                               |
  | --- PUT /sp/0/dev/mfg "acme" --------------->  |
  |                                               |
  |                                               |
  | <-- 2.01 Created ----------------------------  |
  |                                               |
```

Req: PUT /sp/0/dev/mfg
Payload: acme
Res: 2.01 Created

The example below shows how a Sleepy Node can indicate that it is supposed to send a temperature value at least every hour to keep its delegated resource active.

```
Sleepy                                          Proxy
  |                                               |
  | --- PUT /sp/0/sen/temp?lt=3600 "22" -------->  |
  |                                               |
  |                                               |
  | <-- 2.04 Changed ----------------------------  |
  |                                               |
```

Req: PUT /sp/0/sen/temp?lt=3600
Payload: 22
Res: 2.04 Changed

The use of repeated CoAP PUT can be avoided by writing all relevant resources into the Proxy in one operation by means of the Batch interface described in [I-D.ietf-core-interfaces]. After successful initialization, a Proxy SHOULD enable resource discovery for the new delegated resources by updating its /.well-known/core resource.

5.5. Sleepy Node updates delegated resource at Proxy

A Sleepy Node can update a delegated resource at the Proxy (REPORT A) using standard CoAP PUT requests on the delegated resource as shown in Section 5.4.

When a Sleepy Node sends a PUT request to update its resources, the response MAY contain a link-format payload. The payload does not

directly relate to the target resource of the PUT request. Instead, it is a list of web links to resources that have been modified by clients since either the last PUT request or the last call to the modification check interface (see Section 5.6).

5.6. Sleepy Node READs resource updates from Proxy

This function allows a Sleepy Node to retrieve a list of delegated resources that have been modified at the Proxy by other nodes. The interface format for GET is the same as the one specified for PUT in Section 5.4.

A configuring Node (EP) can update a resource in the Proxy. The Sleepy Node receives an indication of the changed resources as specified in Section 5.5.

The Sleepy Node can send GET requests to its Proxy on each delegated resource in order to receive their updated representation. The example in Figure 4 shows a configuration node which changes the name of a Sleepy Node at the Proxy. The Sleepy Node can then check and read the modification in its resource.

```
     Sleepy                  Proxy                     EP
       |                        | <---PUT /sp/0/dev/n----|
       |                        | Payload: Sensor1       |
    Wake-up                     |---2.04 Changed-------->|
     event                      |                        |
       |                        |                        |
       |--POST /sp/0/dev/.. -->|                        |
       |<----2.04 Changed------|                        |
       | Payload: <sp/0/dev/n> |                        |
       |                        |                        |
       |---GET /sp/0/dev/n---->|                        |
       |<-----2.05 Content-----|                        |
       |  Payload: Sensor1     |                        |
       |                        |                        |
```

Figure 4: Example: A Sleepy Node READs resource updates from his Proxy

6. Delegate Interface

This section details the functions belonging to the delegate interface.

6.1. Discovering Endpoint discovers Sleepy Node at Proxy

Through this function, a Discovering Endpoint can discover one or more Sleepy Node(s) at a Proxy. In case a Resource Directory is not present, this is the only way to discover Sleepy Nodes. A CoAP client discovers resources owned by the Sleepy Node but hosted on the Proxy using typical mechanisms such as one or more GETs on the resource /.well-known/core [RFC6690].

Resource discovery between an Endpoint and a proxy or an Endpoint and a RD needs special care to take into account the fact that resources from a Sleepy Node might appear duplicated. EPs SHOULD employ 2-step resource discovery by looking up Sleepy Nodes AND resource types to detect duplicate resources. EPs MAY use single-step resource discovery only if the Sleepy Node can register with no more than one Proxy. An EP can use the "ep" link attribute as a filter on the "/.well-known/core" resource to retrieve a list of endpoints and detect duplicate Sleepy Nodes registered on multiple proxies. An EP can use the "ep" type of lookup to do the same on a RD. The result of endpoint discovery is then used to filter out duplicate resources returned from simple resource discovery.

The following example shows a client discovering the Sleepy Nodes and learning that the Sleepy Node 0224e8fffe925dcf is registered on two Proxies.

```
EP                                        proxy1   proxy2
 |                                          |        |
 | ----- GET /.well-known/core?ep=* ------->|------>|
 |                                          |        |
 |                                          |        |
 | <---- 2.05 Content "</sp/0>..."  --------|        |
 |                                          |        |
 | <---- 2.05 Content "</sp/0>..."  --------|-------|
```

```
Req: GET coap://[ff02::1]/.well-known/core?ep=*
Res: 2.05 Content
</sp/0>;ep="0224e8fffe925dcf"
Res: 2.05 Content
</sp/0>;ep="02004cfffe4f4f50"
</sp/1>;ep="0224e8fffe925dcf"
```

From the previous exchange and the next resource discovery request, the EP can infer that the resources coap://sp1/sp/0/sen/temp and coap://sp2/sp/1/sen/temp actually come from the same Sleepy Node with ep=0224e8fffe925dcf.

```
EP                                          proxy1  proxy2
 |                                             |       |
 | - GET /.well-known/core?rt=ipso:ucum.Cel ->|------>|
 |                                             |       |
 |                                             |       |
 | <---- 2.05 Content "</sp/0>..."  ----------|       |
 |                                             |       |
 | <---- 2.05 Content "</sp/1>..."  ----------|-------|
```

```
Req: GET coap://[ff02::1]/.well-known/core?rt=ucum.Cel
                                        &ep=0224e8fffe925dcf
Res: 2.05 Content
</sp/0/sen/temp;rt="ucum.Cel"
Res: 2.05 Content
</sp/1/sen/temp>;rt="ucum.Cel"
```

6.2. Proxy REPORTs events to Endpoint

This interface can be used by the Endpoint to receive event report message to Proxy (REPORT A) which further notifies it to interested Destination Endpoint(s)(REPORT B). This indirect reporting is useful for a scalable solution, e.g. there may be many interested subscribers but the Sleepy Node itself can only support a limited number of subscribers given its limits on battery energy. A client interested in the events related with a specific resource may send a CoAP GET to the Proxy, to obtain the last published state. If a Reading node is interested in receiving updates whenever the Sleepy Node reports new event to its Proxy, it can use observe [I-D.ietf-core-observe] at the Proxy for that specific resource.

A proxy using the CoAP protocol [RFC7252] SHOULD accept to establish a CoAP observation relationship between the delegated resource and a client as defined in [I-D.ietf-core-observe].

A Sleepy Node may stop updating its delegated resources without explicitly removing its delegated resource (e.g. transition to another proxy after network unreachability detection). An Endpoint can detect this situation when the corresponding delegated resource has expired. Upon receipt of a response with error code 4.04 "Not Found", an Endpoint SHOULD restart resource discovery to determine if the resources are now delegated to another proxy.

The interface function is specified as follows:

Interaction: EP -> Proxy

Method: Defined at registration

URI Template: /{+location}{+resource}

URI Template Variables:

location := This is the Location path returned by the Proxy as a result of a successful registration.

resource := This is the relative path to a delegated resource managed by a Sleepy Node.

Content-Type: Defined at registration

In the example below an EP observes the changes of temperature through the Proxy.

```
Sleepy                        Proxy                       EP
  |                              |                        |
  |                              | <- GET /sp/0/sen/temp - |
  |                              |          (observe)     |
  |                              |                        |
  |                              | -- 2.05 Content "22" -> |
  |                              |                        |
  | - PUT /sp/0/sen/temp "23" -> |                        |
  |                              |                        |
  | <- 2.04 Changed ------------ |                        |
  |                              |                        |
  |                              | -- 2.05 Content "23" -> |
```

6.3. A Node WRITEs to Sleepy Node via Proxy

A Configuring Node uses CoAP PUT to write information (such as configuration data) to the Proxy, where the information is destined for a Sleepy Node. Upon change of a delegated resource, an internal flag is set in the Proxy that the specific resource has changed. Next time the Sleepy Node wakes up, the Sleepy Node checks the Proxy for any modification of its delegated resources and reads those changed resources using CoAP GET requests, as shown in Figure 4. The allowed resources that a Configuring Node can write to, and the CoAP Content-Format of those CoAP resources, is determined in the initial registration phase.

The following example shows a commissioning tool (EP) changing the name of a Sleepy Node through a Proxy. The Sleepy Node detects this change right after updating its current temperature.

```
   Sleepy                      Proxy                       EP
     |                            |                          |
     |                            | <-- PUT /sp/0/dev/n --- |
     |                            |                          |
     |                            | -- 2.04 Changed ------> |
     |                            |                          |
     | - PUT /sp/0/sen/temp --->  |                          |
     | <- 2.04 Changed ---------  |                          |
     | Payload: <sp/0/dev/n> ---  |                          |
     |                            |                          |
     | - GET /sp/0/dev/n ------>  |                          |
     |                            |                          |
     | <- 2.05 Content ---------  |                          |
     |                            |                          |
```

Req: PUT /sp/0/dev/n
Payload: "sensor-1"
Res: 2.04 Changed

Req: PUT /sp/0/sen/temp
Payload: "24"
Res: 2.04 Changed, Content-Type: application/link-format
Payload: "</sp/0/dev/n>"

Req: GET /sp/0/dev/n
Res: 2.05 Content
Payload: "sensor-1"

6.4. A Node READs information from Sleepy Node via Proxy

A Reading Node uses standard CoAP GET to read information of a Sleepy Node via a Proxy. However, not all information/resources from the Sleepy Node may be copied to the Proxy. In that case, the Reading Node cannot get direct access to resources that are not delegated to the Proxy. The strategy to follow in that case is to first WRITE to the Sleepy Node (via the Proxy, Section 6.3) a request for reporting this missing information; where the request can be fulfilled by the Sleepy Node the next time the Sleepy Node wakes up.

7. Direct Interface

This section details the functions belonging to the direct interface.

7.1. Sleepy Node REPORTs events directly to Destination Node

When the Sleepy Node needs to report an event to Destination nodes or groups of Destination nodes present in the subscribers list, it

becomes Awake and then it can use standard CoAP POST unicast or multicast requests to report the event.

TODO: MC example

7.2. A Sleepy Node READs information from a Server Node

A Sleepy Node while Awake uses standard CoAP GET to read any information from a Server Node. While the Sleepy Node awaits a CoAP response containing the requested information, it remains awake. To increase battery life of Sleepy Nodes, such an operation should not be performed frequently.

8. Realization with PubSub broker

The PubSub broker [I-D.koster-core-coap-pubsub] can be used to implement the REPORT function of the Sleepy Node proxy specified in this document. However, there are some differences to be taken into account:

- The PubSub broker handles topics. In the case of the proxy the topics must be equated to resources.

- Clients publish anonymously updates to a topic. In the case of the proxy, a delegated resource is bound to one given node that is allowed to update it. For the same functionality, the PubSub broker must restrict topic updates to one client only. The client linked to the topic must be visible to the clients which subscribe to the topic.

In addition, some other functionality needs to be added to the PubSub broker to satisfy the interaction model shown in Figure 1:

- the READ function from Sleepy Node to proxy is not covered by the PubSub broker. The PubSub broker needs to piggy-back a "check topic" on the confirmation of a publication by the proxy. The proxy can then perform a Read on the signalled topic.

- The interaction "register resources" from proxy to Resource Directory, shown in Figure 3, is not part of the PubSub broker.

9. IANA Considerations

The new Resource Type (rt=) Link Target Attribute, ’core.sp’ needs to be registered in the "Resource Type (rt=) Link Target Attribute Values" sub registry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

10. Security Considerations

For the communication between Sleepy Node and Proxy it MAY be sufficient to use Layer 2 (MAC) security without the recommended use of DTLS. However, it must be ascertained that the Sleepy Node can communicate only with a given secured Proxy. A Sleepy Node may obtain the Layer 2 network key using the bootstrapping mechanism described in [I-D.kumar-6lo-selective-bootstrap]. DTLS MUST be used over link-layer security for further transport-layer protection of messages between Regular Nodes and Proxies in the network. There are no special adaptations needed of the DTLS handshake to support Sleepy Nodes. During the whole handshake, Sleepy Nodes are required to remain awake to avoid that, in case of small retransmission timers, the other node may think the handshake message was lost and starts retransmitting. In view of this, the only key point, therefore, is that DTLS handshakes are not performed frequently to save on battery power. Based on the DTLS authentication, also an authorization method could be implemented so that only authorized nodes can e.g.

- Act as a Proxy for a Sleepy Node. (The Proxy shall be a trusted device given its important role of storing values of parameters for the delegated resources);

- READ data from Sleepy Nodes;

- WRITE data to Sleepy Nodes (via the Proxy);

- Receive REPORTs from Sleepy Nodes (direct or via Proxy).

11. Acknowledgements

Much of the text and examples in this document are copied from [I-D.vial-core-mirror-server]. Matthieu Vial has generously authorized us to use his text. Rahman Akbar has pointed out the CoAP dependency of earlier versions.

12. Changelog

RFC editor, please delete this section before publication.

From version 2 to version 3:

Introduced interfaces and copied examples and text from mirror server draft.

From version 3 to version 4:

Comparison with PubSub Broker completed.

Mistakes in examples removed.

Less dependence on 6LowPAN networks.

Added Design motivation section.

13. References

13.1. Normative References

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

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

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <http://www.rfc-editor.org/info/rfc7390>.

13.2. Informative References

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., and M. Koster, "CoRE Interfaces", draft-ietf-core-interfaces-03 (work in progress), July 2015.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-04 (work in progress), July 2015.

[I-D.koster-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-02 (work in progress), July 2015.

[I-D.kumar-6lo-selective-bootstrap]
Kumar, S. and P. Stok, "Security Bootstrapping over IEEE 802.15.4 in selective order", draft-kumar-6lo-selective-bootstrap-00 (work in progress), March 2015.

[I-D.vial-core-mirror-server]
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-server-01 (work in progress), April 2013.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <http://www.rfc-editor.org/info/rfc6763>.

Authors' Addresses

Teresa Zotti
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
The Netherlands

Phone: +31 6 21175346
Email: teresa.zotti@philips.com

Peter van der Stok
Consultant

Phone: +31 492474673
Email: consultancy@vanderstok.org

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

Phone: +31 6 55408986
Email: esko.dijk@philips.com