Abstract

This draft discusses the need for a consistent messaging layer that can be used but the transport protocols as they adapt to the CoAP Request/Response layer. In addition, this draft provides comments to the TCP transport implementaton described by [I-D.tschofenig-core-coap-tcp-tls].

Status of This Memo

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

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

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

This Internet-Draft will expire on December 19, 2015.

Copyright Notice

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

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

In review of [I-D.tschofenig-core-coap-tcp-tls], we realized that this draft:

- Didn’t support the CoAP message layer’s use of ACK/RST in CON and NON message types or the message-id. In fact, the draft explicitly removed support for the CON message types and didn’t support CoAP ACK mechanisms – relying on the TCP ack/rst/fin messages and timeout mechanisms.

- Didn’t explicitly discuss how piggy backed responses would be handled.

- Made the assumption that the Blockwise protocol was supported but did not describe how Blockwise would be supported within the concep of TCP connections.

- Didn’t explicitly discuss how TCP connections related to the higher layer Request-Response/Observe-Notify and the newer Publish and Subscribe message exchange patterns.

2. Confusion in the CoAP Message Layer

RFC 7252 described a Message Layer to allow for Confirmable/Non-confirmable delivery of Request/Response messages. The unstated purpose of this message layer was that it was to be used for unreliable transports (e.g., UDP, SMS). Several drafts (e.g., Observe [I-D.ietf-core-observe], Block [I-D.ietf-core-block]) and standards groups (e.g., OMA, oneM2M) have referred to the Message Layer (Adaptation Layer) primitives (e.g., CON, NON, ACT, RST, Message id) in their processing. As such the interface between the Application and Request/Response Layer was assumed to extend into the primitives offered by the Adaptation Layer. Subsequent clarifications of the Application Layer interaction was provided that...
Applications (e.g., LWM2M Clients and Servers) interact with the CoRE Application Features and/or the underlying Request/Response Layers.

The following Figure depicts the CoAP Layers with the initial set of Transport protocols and CoAP Features.

While the clarification of the Application Layer provided an explanation of how Applications should interact with the Request/Response Layer, the discussion highlighted an additional problem. There isn’t a single consistent interface between the Adaptation Layer and the Request/Response Layer. This consistency was lost when new Transport protocols did not implement the message primitives (e.g., CON/NON, ACK, RST) of the UDP/SMS Messaging Layer.
The following Figure depicts the CoAP Layers with the new set of Transport protocols.

Figure 2: CoAP Layers (New Transports)

3. Standard Primitives vs Transport Specific Adaptation

If a standard set of primitives were used, each Transport protocol would document how to implement the CON and NON messages with ACK and RST responses. The Request/Response Layer feature would describe how to adapt timeouts and state processing of the Message Layer. This would provide for a clean delineation of responsibility such that developers of new Transport protocols and Request/Response features would know exactly what the behavior is that is provided and consumed by each layer (i.e., Transport, Request/Response).
The following Figure depicts the CoAP Layers with a standard Message Layer.

```
Figure 3: CoAP Layers - Standard Primitives

If transport specific adaptation is used, the transport protocol would specify how the Request/Response layer exchange patterns and features would be adapted by the protocol. This will become very difficult to maintain as each new feature that needs aspects of a transport protocol will have to also include those aspects such as was done in the Observe draft.

The side-effect of losing the standard set of messaging primitives is that each Transport will have to document how that transport adapts to the various elements of the Request/Response Layer (e.g., Block, Observe, Request, Response) rather than document how they would implement the standard set of messaging primitives. In addition each new Request/Response feature will have to document how it will interact with each Transport Layer.
```
The following Figure depicts the CoAP Layers with Adaptation Layers specific to each Transport Protocol.

Another side-effect of not using the existing set of message layer primitives is that Applications MUST be aware of the Transport bearer when invoking requests because they have to set the type of message (CON, NON) because a Transport (e.g., TCP) may not support the message (e.g., CON).

4. Standardize Message Layer

Using the existing CoAP Message Layer as the standard set of primitives allows IETF Drafts that focus on features in the Request/Response Layer to know what is provided by any Transport protocol. Likewise IETF Drafts for CoRE elements will know the messages that are needed to be either implemented or provided.
Note: The draft does not suggest Message Layer mechanisms like transport specific timeout processing will be exposed, just the messaging.
The following Figure depicts the message interactions between the CoAP Layers using a standardized Message Layer.

Application Layer

Request/Response Layer
Request-Response,
Observe-Notify,
Block, Pub-Sub

Confirmable | Non-confirmable

Message Layer
(Transport Adaptation)

Transport Layer
(TCP, UDP, SMS, Websockets)

Figure 5: CoAP Layers - Standard Message Layer
5. Issues with the Current TCP Draft

The current draft [I-D.tschofenig-core-coap-tcp-tls], has the following issues:

1. TCP Connections: TCP connections in the current draft are currently limited to a single Request/Response information exchange. This limitation means that multiple TCP connections are needed for parallel information exchanges. For example, an Observe/Notification information exchange would have to be on a different TCP connection as a simple Get request, causing multiple costly TCP connections to be established. In addition, long lived TCP connections could not be supported unless the Application serialized the Request/Response exchange which is difficult with Request/Response features like Observe. As such, modifications to the draft to allow Long TCP connections with multiple Request/Response Information exchanges is needed.

2. Blockwise Transfer: The current draft does not include documentation of how to handle Block transfers especially with the use of the TCP ack and empty messages. Actually the Blockwise transfer draft should be modified to use the Request/Response terminology instead of the ACK message terms (e.g., Section 3.1 Block2 examples. This is an example of the confusion caused by not having a standardized set of message primitives.

3. Accounting for Request/Response Layer Usage - Observe: The current TCP draft needs to document how to account for:

   * Confirmable messages in the Observe draft (section 1.2, 3.4, 3.6, 4.5, 4.5.1): The use of message ID in non-confirmable messages (section 4.5) and adpatation of congestion control (section 4.5.1). The TCP draft should document how it emulates the behavior of the confirmable messages in each of the sections. For example the use of TCP acks as a replacement for CON message ACKs.

   * Use of Message Id in non-confirmable messages in the Observe draft (section 4.5): Since Message Ids are elided, the draft needs to document how the RST messages for Notifications should be handled unless Message Ids are indeed supported in a future TCP draft.

   * Adaptation of congestion control in the Observe draft (section 4.5.1): The TCP drafts needs to document how congestion control would be done for simultaneous Notifications.
Use of the Message Id to ensure no duplication through the Request/Response Layer: The TCP protocol will only ensure duplication at the TCP layer. The TCP protocol doesn’t prevent an invoking Request/Response layer from sending the message more than once for any reason (good or bad). As such the support of Message ID is still needed as the TCP layer is insufficient because the solution cannot address possibilities at the Request/Response layer.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

None

8. References

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

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

[I-D.tschofenig-core-coap-tcp-tls]

Author’s Address

Timothy Carey
Alcatel-Lucent
TX
US

Phone: +1-972-415-2065
Email: timothy.carey@alcatel-lucent.com
Advanced Guidelines for HTTP-CoAP Mapping Implementations
draft-castellani-core-advanced-http-mapping-06

Abstract

This draft describes advanced features for HTTP-CoAP proxy implementers. It details deployment options, discusses possible approaches for URI mapping, and provides useful considerations related to protocol translation.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2016.

Copyright Notice

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

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

1. Terminology and Conventions .................................. 3
2. Introduction .................................................. 3
3. Use Case: HTTP/IPv4-CoAP/IPv6 Proxy ............................. 3
4. URI Mapping via HTTP Cache Control Extensions .................. 6
5. Multiple Message Exchanges Mapping .............................. 6
   5.1. Relevant Features of Existing Standards ..................... 6
   5.1.1. Multipart Messages ...................................... 6
   5.1.2. Immediate Message Delivery ................................ 7
   5.1.3. Detailing Source Information .............................. 7
   5.2. Multicast Mapping .......................................... 7
   5.2.1. URI Identification and Mapping ........................... 8
   5.2.2. Request Handling ........................................ 8
   5.2.3. Examples ................................................. 9
   5.3. Multicast Response Caching .................................. 11
   5.4. Observe Mapping ............................................ 12
   5.4.1. Identification ........................................... 12
   5.4.2. Notification(s) Mapping ................................... 14
   5.4.3. Examples ................................................ 15
6. HTML5 Scheme Handler Registration ................................ 21
7. Placement and Deployment ......................................... 21
8. Examples ....................................................... 22
9. Acknowledgements ............................................... 24
10. IANA Considerations ............................................ 24
11. Security Considerations ......................................... 24
   11.2. Subscription .............................................. 25
12. References ................................................... 25
   12.1. Normative References ...................................... 25
   12.2. Informative References .................................... 26
Appendix A. Internal Mapping Functions (from an Implementer’s Perspective) .................................. 27
   A.1. URL Map Algorithm ......................................... 28
   A.2. Security Policy Map Algorithm .............................. 29
   A.3. Content-Type Map Algorithm ................................ 30
Authors’ Addresses ............................................... 31
1. Terminology and Conventions

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

This document assumes readers are familiar with the terms and concepts that are used in [I-D.ietf-core-coap] . In addition, this document defines the following terminology:

A device providing cross-protocol HTTP-CoAP mapping is called an HTTP-CoAP cross-protocol proxy (HC proxy).

At least two different kinds of HC proxies exist:

- One-way cross-protocol proxy (1-way proxy): This proxy translates from a client of a protocol to a server of another protocol but not vice-versa.

- Two-way (or bidirectional) cross-protocol proxy (2-way proxy): This proxy translates from a client of both protocols to a server supporting one protocol.

2. Introduction

RESTful protocols, such as HTTP [RFC2616] and CoAP [I-D.ietf-core-coap], can interoperate through an intermediary proxy which performs cross-protocol mapping.

A base reference for the mapping process is provided in [I-D.ietf-core-coap]. However, depending on the involved application, deployment scenario, or network topology, such mapping can be realized using a wide range of intermediaries.

Moreover, the process of implementing such a proxy can be complex, and details regarding its internal procedures and design choices deserve further discussion, which is provided in this document.

This draft itself is an evolution of the mapping features covered in [I-D.ietf-core-http-mapping].

3. Use Case: HTTP/IPv4-CoAP/IPv6 Proxy

This section covers the expected common use case regarding an HTTP/IPv4 client accessing a CoAP/IPv6 resource.

While HTTP and IPv4 are today widely adopted communication protocols in the Internet, a pervasive deployment of constrained nodes...
exploiting the IPv6 address space is expected: enabling direct interoperability of such technologies is a valuable goal.

An HC proxy supporting IPv4/IPv6 mapping is said to be a v4/v6 proxy.

An HC v4/v6 proxy SHOULD always try to resolve the URI authority, and SHOULD prefer using the IPv6 resolution if available. The authority part of the URI is used internally by the HC proxy and SHOULD NOT be mapped to CoAP.

Figure 1 shows an HTTP client on IPv4 (C) accessing a CoAP server on IPv6 (S) through an HC proxy on IPv4/IPv6 (P). The DNS has an A record for "node.coap.something.net" resolving to the IPv4 address of the HC proxy, and an AAAA record with the IPv6 address of the CoAP server.
The proposed example shows the HC proxy operating also the mapping between IPv4 to IPv6 using the authority information available in any HTTP 1.1 request. This way, IPv6 connectivity is not required at the HTTP client when accessing a CoAP server over IPv6 only, which is a typical expected use case.

When P is an interception HC proxy, the CoAP request SHOULD have the IPv6 address of C as source (IPv4 can always be mapped into IPv6).

---

<table>
<thead>
<tr>
<th>Source: IPv4 of C</th>
<th>Destination: IPv4 of P</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /foo HTTP/1.1</td>
<td>Host: node.coap.something.net</td>
</tr>
<tr>
<td>..other HTTP headers ..</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source: IPv6 of P</th>
<th>Destination: IPv6 of S</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON GET</td>
<td>URI-Path: foo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source: IPv6 of S</th>
<th>Destination: IPv6 of P</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON 2.00</td>
<td>&quot;bar&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source: IPv6 of P</th>
<th>Destination: IPv6 of S</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source: IPv4 of P</th>
<th>Destination: IPv4 of C</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP/1.1 200 OK</td>
<td>.. other HTTP headers ..</td>
</tr>
</tbody>
</table>

| bar |

---

Figure 1: HTTP/IPv4 to CoAP/IPv6 Mapping
The described solution takes into account only the HTTP/IPv4 clients accessing CoAP/IPv6 servers; this solution does not provide a full fledged mapping from HTTP to CoAP.

In order to obtain a working deployment for HTTP/IPv6 clients, a different HC proxy access method may be required, or Internet AAAA records should not point to the node anymore (the HC proxy should use a different DNS database pointing to the node).

When an HC interception proxy deployment is used this solution is fully working even with HTTP/IPv6 clients.

4. URI Mapping via HTTP Cache Control Extensions

An advanced strategy for triggering the cross-proxy that a translation is needed can be done via the HTTP Cache Control Extensions described in Section 5.2.3 of [RFC7234]. Specifically two new extensions can be defined, i.e. cross-coap and cross-coaps, that when included in a request to an HC forward cross-proxy translate the request to coap or coaps.

5. Multiple Message Exchanges Mapping

This section discusses the mapping of the multicast and observe features of CoAP, which have no corresponding primitive in HTTP, and as such are not immediately translatable.

The mapping, which must be considered in both the arrow directions (H->C, C->H) may involve multi-part responses, as in the multicast use case, asynchronous delivery through HTTP bidirectional techniques, and HTTP Web Linking in order to reduce the semantics lost in the translation.

5.1. Relevant Features of Existing Standards

Various features provided by existing standards are useful to efficiently represent sessions involving multiple messages.

5.1.1. Multipart Messages

In particular, the "multipart/*" media type, defined in Section 5.1 of [RFC2046], is a suitable solution to deliver multiple CoAP responses within a single HTTP payload. Each part of a multipart entity SHOULD be represented using "message/http" media type containing the full mapping of a single CoAP response as previously described.
5.1.2. Immediate Message Delivery

An HC proxy may prefer to transfer each CoAP response immediately after its reception. This is possible thanks to the HTTP Transfer-Encoding "chunked", that enables transferring single responses without any further delay.

A detailed discussion on the use of chunked Transfer-Encoding to stream data over HTTP can be found in [RFC6202]. Large delays between chunks can lead the HTTP session to timeout, more details on this issue can be found in [I-D.thomson-hybi-http-timeout].

An HC proxy MAY prefer (e.g. to avoid buffering) to transfer each response related to a multicast request as soon as it comes in from the server. One possible way to achieve this result is using the "chunked" Transfer-Encoding in the HTTP response, to push individual responses until some trigger is fired (timeout, max number of messages, etc.).

An example showing immediate delivery of CoAP responses using HTTP chunks will be provided in Section 5.4, while describing its application to an observe session.

5.1.3. Detailing Source Information

Under some circumstances, responses may come from different sources (i.e. responses to a multicast request); in this case details about the actual source of each CoAP response MAY be provided to the client. Source information can be represented using HTTP Web Linking as defined in [RFC5988], by adding the actual source URI into each response using Link option with "via" relation type.

5.2. Multicast Mapping

In order to establish a multicast communication such a feature should be offered either by the network (i.e. IP multicast, link-layer multicast, etc.) or by a gateway (i.e. the HC proxy). Rationale on the methods available to obtain such a feature is out-of-scope of this document, and extensive discussion of group communication techniques is available in [I-D.ietf-core-groupcomm].

Additional considerations related to handling multicast requests mapping are detailed in the following sections.
5.2.1. URI Identification and Mapping

In order to successfully handle a multicast request, the HC proxy MUST successfully perform the following tasks on the URI:

Identification: The HC proxy MUST understand whether the requested URI identifies a group of nodes.

Mapping: The HC proxy MUST know how to distribute the multicast request to involved servers; this process is specific of the group communication technology used.

When using IPv6 multicast paired with DNS, the mapping to IPv6 multicast is simply done using DNS resolution. If the group management is performed at the proxy, the URI or part of it (i.e. the authority) can be mapped using some static or dynamic table available at the HC proxy. In Section 3.5 of [I-D.ietf-core-groupcomm] discusses a method to build and maintain a local table of multicast authorities.

5.2.2. Request Handling

When the HC proxy receives a request to a URI that has been successfully identified and mapped to a group of nodes, it SHOULD start a multicast proxying operation, if supported by the proxy.

Multicast request handling consists of the following steps:

Multicast TX: The HC proxy sends out the request on the CoAP side by using the methods offered by the specific group communication technology used in the constrained network;

Collecting RXs: The HC proxy collects every response related to the request;

Timeout: The HC proxy has to pay special attention in multicast timing, detailed discussion about timing depends upon the particular group communication technology used;

Distributing RXs to the client: The HC proxy can distribute the responses in two different ways: batch delivering them at the end of the process or on timeout, or immediately delivering them as they are available. Batch requires more caching and introduces delays but may lead to lower TCP overhead and simpler processing. Immediate delivery is the converse. A trade-off solution of partial batch delivery may also be feasible and efficient in some circumstances.
5.2.3. Examples

Figure 2 shows an HTTP client (C) requesting the resource "/foo" to a group of CoAP servers (S1/S2/S3) through an HC proxy (P) which uses IP multicast to send the corresponding CoAP request.
Figure 2: Unicast HTTP to Multicast CoAP Mapping

Code:

```
+----->      |     |     |     |
     |     |     |     |
     +---->|     |     |     |  GET /foo HTTP/1.1
        |     |     |     |  Host: group-of-nodes.coap.something.net
        |     |     |     |  .. other HTTP headers ..
<--------+     |     |     |     |  NON GET
        |     |     |     |  URI-Path: foo
X--------+     |     |     |     |  NON 2.00
        |     |     |     |  "S2"
<------+     |     |     |     |  NON 2.00
        |     |     |     |  "S3"
<-----+     |     |     |     |  "S1"
        |     |     |     |  ... Timeout ...
<-----+     |     |     |     |  HTTP/1.1 200 OK
        |     |     |     |  Content-Type: multipart/mixed;
        |     |     |     |       boundary="response"
        |     |     |     |  .. other HTTP headers ..
        |     |     |     |  --response
        |     |     |     |  Content-Type: message/http
        |     |     |     |  HTTP/1.1 200 OK
        |     |     |     |  Link: <http://node2.coap.something.net/foo>; rel=via
        |     |     |     |  S2
        |     |     |     |  --response
        |     |     |     |  Content-Type: message/http
        |     |     |     |  HTTP/1.1 200 OK
        |     |     |     |  Link: <http://node1.coap.something.net/foo>; rel=via
        |     |     |     |  S1
        |     |     |     |  --response--
```
The example proposed in the above diagram does not make any assumption on which underlying group communication technology is available in the constrained network. Some detailed discussion is provided about it along the following lines.

C makes a GET request to group-of-nodes.coap.something.net. This domain name MAY either resolve to the address of P, or to the IPv6 multicast address of the nodes (if IP multicast is supported and P is an interception proxy), or the proxy P is specifically known by the client that sends this request to it.

To successfully start multicast proxying operation, the HC proxy MUST know that the destination URI involves a group of CoAP servers, e.g. the authority group-of-nodes.coap.something.net is known to identify a group of nodes either by using an internal lookup table, using DNS paired with IPv6 multicast, or by using some other special technique.

A specific implementation option is proposed to further explain the proposed example. Assume that DNS is configured such that all subdomain queries to coap.something.net, such as group-of-nodes.coap.something.net, resolve to the address of P. P performs the HC URI mapping by removing the 'coap' subdomain from the authority and by switching the scheme from 'http' to 'coap' (result: "coap://group-of-node.something.net/foo"); "group-of-nodes.something.net" is resolved to an IPv6 multicast address to which S1, S2 and S3 belong. The proxy handles this request as multicast and sends the request "GET /foo" to the multicast group.

5.3. Multicast Response Caching

We call perfect caching when the proxy uses only the cached representations to provide a response to the HTTP client. In the case of a multicast CoAP request, perfect caching is not adequate. This section updates the general caching and congestion control guidelines with specific guidelines for the multicast use case.

Due to the inherent unreliable nature of the NON messages involved and since nodes may have dynamic membership in multicast groups, responding only with previously cached responses without issuing a new multicast request is not recommended. This perfect caching behaviour leads to miss responses of nodes that later joined the multicast group, and/or to repeatedly serve partial representations due to message losses. Therefore a multicast CoAP request SHOULD be sent by a HC proxy for each incoming request addressed to a multicast group.

Caching of multicast responses is still a valuable goal to pursue reduce network congestion, battery consumption and response latency.
Some considerations to be performed when adopting a multicast caching behaviour are outlined in the following paragraph.

Caching of multicast GET responses MAY be implemented by adopting some technique that takes into account either knowledge about dynamic characteristics of group membership (occurrence or frequency of group changes) or even better its full knowledge (list of nodes currently part of the group).

When using a technique exploiting this knowledge, valid cached responses SHOULD be served from cache.

5.4. Observe Mapping

By design, and certainly not without a good rationale, HTTP lacks a publish-subscriber facility. This implies that the mapping of the CoAP observe semantics has to be created ad hoc, perhaps by making use of one of the well-known HTTP techniques currently employed to establish an HTTP bidirectional connection with the target resource – as documented in [RFC6202].

In the following sections we will describe some of the approaches that can be used to identify an observable resource and to create the communication bridging needed to set up an end to end HTTP-CoAP observation.

5.4.1. Identification

In order to appropriately process an observe request, the HC proxy needs to know whether a given request is intended to establish an observation on the target resource, instead of triggering a regular request-response exchange.

At least two different approaches to identify such special requests exist, as discussed below.

5.4.1.1. Observable URI Mapping

An URI is said to be observable whenever every request to it implicitly requires the establishment of an HTTP bidirectional connection to the resource.

Such subscription to the resource is always paired, if possible, to a CoAP observe session to the actual resource being observed. In general, multiple connections that are active with a single observable resource at the same time, are multiplexed to the single observe session opened by the intermediary. Its notifications are then de-multiplexed by the HC proxy to every HTTP subscriber.
An intermediary MAY pair a couple of distinct HTTP URIs to a single CoAP observable resource: one providing the usual request-response mediated access to the resource, and the other that always triggers a CoAP observe session.

5.4.1.1.1. Discovery

As shown in Figure 3, in order to know whether an URI is observable, an HTTP UA MAY do a pre-flight request to the target resource using the HTTP OPTIONS method (see section 6.2 of [I-D.ietf-httpbis-p2-semantics]) to discover the communication options available for that resource.

If the resource supports observation, the proxy adds a Link Header [RFC5988] with the "obs" attribute as link-param (see Section 7 of [I-D.ietf-core-observe]).

```
C       P       S
|       |       |  OPTIONS /kitchen/temp HTTP/1.1
|-------|       |  Host: node.coap.something.net
|       |-------|  CON GET
|       |  Uri-Path: /.well-known/core?anchor=/kitchen/temp
|<------|  ACK 2.05
|       |  Payload: </kitchen/temp>;obs
|-------|  HTTP/1.1 200 OK
|       |  Link: </kitchen/temp>; obs; type="application/atom+xml"
|       |  Allow: GET, OPTIONS
```

Figure 3: Discover Observability with HTTP OPTIONS

5.4.1.2. Differentiation Using HTTP Header

Discerning an observation request through in-protocol means, e.g. via the presence and values of some HTTP metadata, avoids introducing static "observable" URIs in the HC proxy namespace. Though ideally the former should be preferred, there seems to be no standard way to use one of the established HTTP headers to convey the observe semantics.

Standardizing such methods is out-of-scope of this document, so we just point out some possible approaches that in the future may be used to differentiate observation requests from regular requests.

5.4.1.2.1. Expect Header

The first method involves the use of the Expect header as defined in Section 9.3 of [I-D.ietf-httpbis-p2-semantics]. Whenever an HC proxy receives a request with a "206-partial-content" expectation, the proxy MUST fulfill this expectation by pairing this request to either a new or existing observe session to the resource.

If the proxy is unable to observe the resource, or if the observation establishment fails, the proxy MUST reply to the client with "417 Expectation Failed" status code.

Given that the Expect header is processed hop-by-hop, this method will fail immediately in case a proxy not supporting this expectation is traversed. For this reason, at present, the said approach can’t be used in the public Internet.

5.4.1.2.2. Prefer Header

A second, very similar, approach involves the use of the Prefer header, defined in [I-D.snell-http-prefer]. The HTTP user agent expresses the preference to establish an observation with the target resource by including a "streaming" preference to request an HTTP Streaming session, or a "long-polling" preference to signal to the proxy its intended polling behaviour (see [RFC6202]).

A compliant HC proxy will try to fulfill the preference, and manifest observation establishment success by responding with a status code of "206 Partial Content". The observation request fails, falling back to a single response, whenever the status code is different from 206.

This approach will never fail immediately, differently from the previous one, even across a chain of unaware proxies; however, as documented in [RFC6202], caching intermediaries may interfere, delay or block the HTTP bidirectional connection, making this approach unacceptable when no weak consistency of the resource can be tolerated by the requesting UA.

5.4.2. Notification(s) Mapping

Multiplexing notifications using a single HTTP bidirectional session needs some further considerations about the selection of the media type that best fits this specific use case.

The usage of two different content-types that are suitable for carrying multiple notifications in a single session, is discussed in the following sections.
5.4.2.1. Multipart Messaging

As already discussed in Section 5.1.1 for multicasting, the "multipart/*" media type is a suitable solution to deliver multiple CoAP notifications within a single HTTP payload.

As in the multicast case, each part of the multipart entity MAY be represented using a "message/http" media type, containing the full mapping of the single CoAP notification mapped, so that CoAP envelope information are preserved (e.g. the response code).

A more sophisticated mapping could use multipart/mixed with native or translated media type.

5.4.2.2. Using ATOM Feeds

Popular observable resources with refresh rates higher than a couple of seconds may be treated as Atom feeds [RFC4287], especially with delay tolerant user agents and where persistence is required.

Figure 3 shows a resource supporting ‘application/atom+xml’ media-type. In such case clients can listen to update notification by regularly polling the resource via opportuneely spaced GETs, i.e. driven by the advertised max-age value.

5.4.3. Examples

Figure 4 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

C manifests its intention to observe T by including the Expect Header in the request; if P or S do not support this interaction, the request MUST fail with "417 Expectation Failed" return code. In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" return code.

At every notification corresponds the emission of a HTTP chunk containing a single part, which contains a "message/http" payload containing the full mapping of the notification. When the observation is dropped by the CoAP server, the HTTP streaming session is closed.
Host: node.coap.something.net
Expect: 206-partial-content
Accept: multipart/mixed

--->
CON GET
Uri-Path: temperature
Observe: 0

<--
ACK 2.05
Observe: 3482
"22.1 C"

HTTP/1.1 206 Partial Content
Content-Type: multipart/mixed; boundary=notification

XX
--notification
Content-Type: message/http

HTTP/1.1 200 OK

22.1 C

... about 60 seconds have passed ...

<--
NON 2.05
Observe: 3542
"21.6 C"

YY
--notification
Content-Type: message/http

HTTP/1.1 200 OK

21.6 C

... if the server drops the relationship ...

<--
NON 2.05
"21.8 C"

ZZ
--notification
Content-Type: message/http

HTTP/1.1 200 OK
Figure 4: HTTP Streaming to CoAP Observe

Figure 5 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

C manifests its intention to observe T by including the Prefer Header in the request; if P or S do not support this interaction, the request silently fails if a status code "200 OK" is returned, which means that no further notification is expected on that session.

In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" status code. At every notification a new response is sent to the pending client, always containing the "206 Partial Content" status code, to indicate that the observe session is still active, so that C can issue a new long-polling request immediately after this notification.

If the observation relationship is dropped by S, P notifies the last received content using the "200 OK" status code, indicating that no further notification is expected on this observe session.
Figure 5: HTTP Long Polling to CoAP Observe
Figure 6 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "kitchen/temp" (T) available on S.

It is assumed that the HC proxy knows that the requested resource is observable (since perhaps being asked beforehand to discover its properties as described in Figure 3.) When asked by the HTTP client to retrieve the resource, it requests an observation - in case it weren’t already in place - and then sends the collected data to the client as an Atom feed. The data coming through in the constrained network is stored locally on the proxy, and forwarded when further requests are received on the HTTP side. As already said, using the Atom format has two main advantages: first, there is always a "current" feed, but there may also be a complete log made available to HTTP clients; secondly, the HTTP intermediaries can play a substantial role in absorbing a fair amount of the load on the HC proxy. The latter is a very important property when the requested resource is or becomes very popular.
Figure 6: Observation via Atom feeds
6. HTML5 Scheme Handler Registration

The draft HTML5 standard offers a mechanism that allows an HTTP user agent to register a custom scheme handler through an HTML5 web page. This feature permits an HC proxy to be registered as "handler" for URIs with the 'web+coap' or 'web+coaps' schemes using an HTML5 web page which embeds the custom scheme handler registration call registerProtocolHandler() described in Section 6.5.1.2 of [W3C.HTML5].

Example: the HTML5 homepage of a HC proxy at h2c.example.org could include the method call:

registerProtocolHandler('web+coap','proxy?url=%s','example HC proxy')

This registration call will prompt the HTTP user agent to ask for the user’s permission to register the HC proxy as a handler for all 'web+coap' URIs. If the user accepts, whenever a 'web+coap' link is requested, the request will be fulfilled through the HC proxy: URI "web+coap://foo.org/a" will be transformed into URI "http://h2c.example.org/proxy?url=web+coap://foo.org/a".

7. Placement and Deployment

In typical scenarios, for communication from a CoAP client to an HTTP origin server, the HC proxy is expected to be located on the client-side (CS). Specifically, the HC proxy is expected to be deployed at the edge of the constrained network as shown in Figure 7.

The arguments supporting CS placement are as follows:

Client/Proxy/Network configuration overhead: CoAP clients require either static proxy configuration or proxy discovery support. This overhead is simplified if the proxy is placed on the same network domain of the client.

TCP/UDP: Translation between CoAP and HTTP requires also UDP to TCP mapping; UDP performance over the unconstrained Internet may not be adequate. In order to minimize the number of required retransmissions on the constrained part of the network and the overall reliability, TCP/UDP conversion SHOULD be performed as soon as possible in the network path.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, thus a CS placement, collecting all the traffic, is strategic for this need.
8. Examples

Figure 8 shows an example implementation of a basic CoAP GET request with an HTTP URI as the value of a Proxy-URI option. The proxy retrieves a representation of the target resource from the HTTP origin server. It converts the payload to a UTF-8 charset, calculates the Max-Age Option from the Expires header field, and derives an entity-tag from the ETag header field.
The example in Figure 9 builds on the previous example and shows an implementation of a GET request that includes a previously returned ETag Option. The proxy makes a Conditional Request to the HTTP origin server by including an If-None-Match header field in the HTTP GET Request. The CoAP response indicates that the response stored by the client is fresh. It includes a Max-Age Option calculated from the HTTP response’s Expires header field.

Figure 8: A Basic CoAP-HTTP GET Request

The example in Figure 9 builds on the previous example and shows an implementation of a GET request that includes a previously returned ETag Option. The proxy makes a Conditional Request to the HTTP origin server by including an If-None-Match header field in the HTTP GET Request. The CoAP response indicates that the response stored by the client is fresh. It includes a Max-Age Option calculated from the HTTP response’s Expires header field.

Figure 8: A Basic CoAP-HTTP GET Request
Figure 9: A CoAP-HTTP GET Request with an ETag Option

9. Acknowledgements

TBD.

10. IANA Considerations

This memo includes no request to IANA.

11. Security Considerations

At the moment of this writing, CoAP and HTTP are missing any cross-protocol security policy mapping.

The HC proxy SHOULD flexibly support security policies between the two protocols, possibly as part of the HC URI mapping function, in order to statically map HTTP and CoAP security policies at the proxy (see Appendix A.2 for an example.)

11.2. Subscription

As noted in Section 7 of [I-D.ietf-core-observe], when using the observe pattern, an attacker could easily impose resource exhaustion on a naive server who’s indiscriminately accepting observer relationships establishment from clients. The converse of this problem is also present, a malicious client may also target the HC proxy itself, by trying to exhaust the HTTP connection limit of the proxy by opening multiple subscriptions to some CoAP resource.

Effective strategies to reduce success of such a DoS on the HTTP side (by forcing prior identification of the HTTP client via usual web authentication mechanisms), must always be weighted against an acceptable level of usability of the exposed CoAP resources.

12. References

12.1. Normative References

[I-D.ietf-core-block]

[I-D.ietf-core-coap]

[I-D.ietf-core-groupcomm]

[I-D.ietf-core-http-mapping]
12.2. Informative References

[I-D.bormann-core-simple-server-discovery]
Bormann, C., "CoRE Simple Server Discovery", draft-
bormann-core-simple-server-discovery-01 (work in progress), March 2012.
Appendix A. Internal Mapping Functions (from an Implementer’s Perspective)

At least three mapping functions have been identified, which take place at different stages of the HC proxy processing chain, involving the URL, Content-Type and Security Policy translation.

All these maps are required to have at least URL granularity so that, in principle, each and every requested URL may be treated as an independent mapping source.

In the following, the said map functions are characterized via their expected input and output, and a simple, yet sufficiently rich, configuration syntax is suggested.
In the spirit of a document providing implementation guidance, the specification of a map grammar aims at putting the basis for a reusable software component (e.g. a stand-alone C library) that many different proxy implementations can link to, and benefit from.

A.1. URL Map Algorithm

In case the HC proxy is a reverse proxy, i.e. it acts as the origin server in face of the served network, the URL of the resource requested by its clients (perhaps having an ‘http’ scheme) shall be mapped to the real resource origin (perhaps in the ‘coap’ scheme).

In case HC is a forward proxy, no URL translation is needed since the client already knows the "real name" of the resource.

An interception HC proxy, instead, MAY use the homogeneous mapping strategy to operate without any pre-configuration need.

As noted in Appendix B of [RFC3986] any correctly formatted URL can be matched by a POSIX regular expression. By leveraging on this property, we suggest a syntax that describes the URL mapping in terms of substituting the regex-matching portions of the requested URL into the mapped URL template.

E.g.: given the source regular expression ‘^http://example.com/coap/.*$’ and destination template ‘coap://$1’ (where $1 stands for the first – and only in this specific case – substring matched by the regex pattern in the source), the input URL "http://example.com/coap/node1/resource2" translates to "coap://node1/resource2".

This is a well established technique used in many today’s web components (e.g. Django URL dispatcher, Apache mod_rewrite, etc.), which provides a compact and powerful engine to implement what essentially is an URL rewrite function.
INPUT
  * requested URL

OUTPUT
  * target URL

SYNTAX
  url_map [rule name] {
    requested_url   <regex>
    mapped_url      <regex match subst template>
  }

EXAMPLE 1
  url_map homogeneous {
    requested_url   'http://.*$
    mapped_url      'coap//$1'
  }

EXAMPLE 2
  url_map embedded {
    requested_url   'http://example.com/coap/.*$
    mapped_url      'coap//$1'
  }

Note that many different url_map records may be given in order to build the whole mapping function. Each of these records can be queried (in some predefined order) by the HC proxy until a match is found, or the list is exhausted. In the latter case, depending on the mapping policy (only internal, internal then external, etc.) the original request can be refused, or the same mapping query is forwarded to one or more external URL mapping components.

A.2. Security Policy Map Algorithm

In case the "incoming" URL has been successfully translated, the HC proxy must lookup the security policy, if any, that needs to be applied to the request/response transaction carried on the "outgoing" leg.
A.3. Content-Type Map Algorithm

In case a set of destination URLs is known as being limited in handling a narrow subset of mime types, a content-type map can be configured in order to let the HC proxy transparently handle the compatible/lossless format translation.

INPUT
* destination URL (after URL map has been applied)
* original content-type

OUTPUT
* mapped content-type

SYNTAX
ct_map {
  target_url  <regex>                 -- one or more targetURLs
  ct_switch   <source_ct, dest_ct>    -- one or more CTs
}

EXAMPLE
ct_map {
  target_url  '^coap://class-1-device/.*$'
  ct_switch   */xml  application/exi
}
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
KoanLogic
Via di Sabbiuno 11/5
Bologna 40136
Italy
Phone: +39 051 644 82 68
Email: tho@koanlogic.com

Esko Dijk
Philips Research
Email: esko.dijk@philips.com
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

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

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

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

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
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.
The combination of the two requirements above, together with text in Section 3 of [RFC6066] which only allows FQDN hostname of the server in the ServerName field, basically binds Certificate Mode DTLS to either DNS, or static host tables containing FQDN’s mappings, or some other system for lookup of registered names which is able to fully mimic the DNS naming scheme.

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

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

1.1. Challenges

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

1. what identifiers should be used in the certificate?

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

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

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

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

2. Terminology and Requirements Language

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

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

3. Resource Directory Names and Domains

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

3.1. Uniqueness Guarantee

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

3.2. Authority Format

3.2.1. Requirements

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

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

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

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

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

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

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

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

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

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

3.2.4. Uri-Host and Uri-Port Considerations

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

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

3.3. SNI Name Type and Server Name Syntax

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

```c
 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:

```c
 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’:

\[
\text{id-rdauthority-san OBJECT IDENTIFIER ::= \{ id-coap 2 \}}
\]

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

4. Client Behaviour

1) Send extended ClientHello containing:
   a) server_name extension with one (and one only) ServerName, case-insensitive matching the authority of the URI to be requested;
   b) Any other potentially useful extension, e.g. client_certificate_url;

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

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

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

5. Server Behaviour

1) Server receives extended ClientHello carrying a server_name extension, and uses the given server_name (with a rd_authority NameType) to select the appropriate certificate. The selected certificate MUST include a SAN otherName with an id-rdauthority-san type-id and value, which MUST case-insensitive match the requested ServerName;
   a) If no certificate can be selected, the server MUST terminate the handshake by sending a fatal-level unrecognized_name(112) alert. [[CREF2: Prefer a single, hard failure, path over soft failure, or worse: ignoring the error altogether.]]
Rationale: do not waste time/energy; provide clear and prompt diagnostic to the peer. It doesn’t look like the condition that could be exploited by a timing attack.]

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

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

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

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

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

6. IANA Considerations

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

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

7. Security Considerations

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

8. Acknowledgements

TODO

9. References

9.1. Normative References


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

[I-D.ietf-dice-profile]


9.2. Informative References

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

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

Authors' Addresses

Thomas Fossati  
Alcatel-Lucent  
3 Ely Road  
Milton, Cambridge  CB24 6DD  
Great Britain  
Email: thomas.fossati@alcatel-lucent.com

Hannes Tschofenig  
ARM Ltd.  
110 Fulbourn Rd  
Cambridge  CB1 9NJ  
Great Britain  
Email: Hannes.tschofenig@gmx.net  
URI:   http://www.tschofenig.priv.at
CoRE Application Descriptions
draft-hartke-core-apps-01

Abstract

The interfaces of RESTful, hypertext-driven applications consist of reusable, self-descriptive components such as Internet media types and link relation types. This document defines a template that application designers can use to describe their application’s interface in a structured way so that other parties can develop interoperable clients and servers or reuse the components in their own applications.

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 25, 2015.

Copyright Notice

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

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

The Constrained Application Protocol (CoAP) [RFC7252] is designed to enable applications implementing the REST architectural style [REST] in Constrained-Node Networks [RFC7228].

As CoAP applications are implemented and deployed, it becomes increasingly important to be able to describe them in some structured way in order to promote interoperability and reuse. Previous efforts like WADL [WADL] however focus on code generation from machine-readable service descriptions and are not truly RESTful [DWNWADL].

REST application interfaces (APIs) are by definition hypertext-driven [RESTAPI]. This means that the interface is a description of the common vocabulary between the client and the server, centered around Internet media types and link relation types, rather than a static list of resources and the operations supported on them.

RESTful applications are often easy to understand, but require some design effort. This is because application designers do not only have to take current requirements into consideration, but also anticipate changes that may be required in the future. The reward is long-term stability and evolvability ("design for decades"). REST is intended for long-lived network-based applications that span multiple organizations [RESTAPI].
This document defines a template for describing the interface of RESTful, hypertext-driven applications in Constrained RESTful Environments (CoRE).

2. Application Descriptions

In this specification, an application description is a named set of reusable, self-descriptive components. It is comprised of:

- URI schemes that identify communication protocols,
- Internet media types that identify representation formats,
- link relation types, and
- optionally, well-known locations.

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

2.1. Communication Protocols

The foundation of a hypertext-driven REST API are the communication protocol(s) spoken between a client and a server. Although HTTP/1.1 [RFC7230] is by far the most common communication protocol for REST APIs, a REST API should typically not be dependent on any specific communication protocol.

2.2. URI Schemes

The use of a particular protocol is guided by URI schemes [RFC7595] that describe the syntax and semantics of URI references found in links and forms (Section 2.4).

A URI scheme refers to a family of protocols, typically distinguished by a version number. For example, the "http" URI scheme refers to the three members of the HTTP family of protocols: HTTP/1.0 [RFC1945], HTTP/1.1 [RFC7230], and HTTP/2 [RFC7540]. The specific HTTP version is negotiated between the client and the server through version indicators in the protocol or the TLS application-layer protocol negotiation (ALPN) extension [RFC7301].

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

One of the most important aspect of hypertext-driven communications is the concept of media types [RFC6838]. Media types are used to label representations so that it is known how the representation should be interpreted, and how it is encoded. The core of an application description should be one or more hypertext media types.

A media type identifies a versioned series of representation formats (Section 2.4): a media type does not identify a particular version of a representation format; rather, the media type identifies the family, and includes provisions for version indicator(s) embedded in the representations themselves to determine more precisely the nature of how the data is to be interpreted. A new media type is only needed to designate a completely incompatible format [MIMEWEB].

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

Media types can be further refined by structured type name suffixes (e.g., "+xml" appended to the base subtype name; see Section 4.2.8 of RFC 6838), or by subtype information embedded in the representations themselves (e.g., "xmlns" declarations in XML documents [XMLNS]). Structured type name suffixes should be preferred, because embedded subtype information cannot be negotiated (e.g., using the CoAP Accept option).

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

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

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

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

2.4. Representation Formats

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

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

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

Representation formats enable hypertext-driven applications when they support the expression of links and/or forms:

- A _link_ is the primary means for a client to change application state. It is a typed connection between two resources [RFC5988] and is comprised of a context (usually the current resource), a link relation type (Section 2.5), a target resource URI, and, optionally, some attributes that describe the link target.

- An _embedding link_ is a link with the additional hint that it, when processed, should be substituted with a representation of the referenced resource. Thus, traversing an embedding link adds to the application state, rather than replacing it.

- A _templated link_ is a link where the client constructs the target resource URI from provided in-band instructions. The specific rules for such instructions are described by the representation format. URI Templates [RFC6570] provide a generic way to construct URIs through variable expansion.

- A _form_ is the primary means for a client to change resource state. It is comprised of a target resource URI, a submission method (PUT, POST, PATCH, or DELETE), and a description of a representation that the service accepts as part of form submission. This description can be a set of form fields, or simply a list of acceptable media types.

- (A form with a submission method of GET is strictly speaking a templated link, since it provides a way to construct a URI and does not change resource state.)
2.5. Link Relation Types

A link relation type identifies the semantics of a link [RFC5988]. For example, a link with the relation type "copyright" indicates that the resource identified by the target URI is a statement of the copyright terms applying to the current context.

Relation types are not to be confused with media types [RFC6838]; they do not identify the format of the representation that results when the link is dereferenced. Rather, they only describe how the current context is related to another resource.

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

2.6. Well-Known Locations

Some applications may require the discovery of information about a host ("site-wide metadata"). For example, [RFC6415] defines a metadata document format for describing hosts; similarly, [RFC6690] defines a link format for the discovery of resources hosted by a server.

Applications that need to define a resource for site-wide metadata can register new "well-known locations". [RFC5785] defines a path prefix in "http" and "https" URIs for this purpose, "/.well-known/"; [RFC7252] extends this concept to "coap" and "coaps" URIs.

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

2.7. URI Structures

Application descriptions must not constrain URI structures in ways that aren’t explicitly allowed by [RFC3986]. In particular, mandating particular forms of URI substructure is inappropriate. [RFC7320] describes this problematic practice and provides some acceptable alternatives for use in application descriptions.

3. Template

Application name:

URI schemes:

Media types:

Link relations:
4. Security Considerations

The security considerations of [RFC3986], [RFC5785], [RFC5988], [RFC6570], [RFC6838], [RFC7320], and [RFC7595] are inherited.

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

5. IANA Considerations

This document includes no request to IANA.

6. Acknowledgements

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

Some of the text in this document has been borrowed from [RESTAPI], [RFC5988], [RFC7320], and [MIMEWEB]. All errors are my own.

This work was funded in part by Nokia.

7. References

7.1. Normative References


7.2. Informative References


[WEBAPIS] Richardson, L. and M. Amundsen, "RESTful Web APIs", O'Reilly, September 2013.


Author’s Address

Klaus Hartke
Universitaet Bremen T2I
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
EMail: hartke@tzi.org
CoAP is a RESTful transfer protocol for constrained nodes and networks. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications will need to transfer larger payloads -- for instance, for firmware updates. With HTTP, TCP does the grunt work of slicing large payloads up into multiple packets and ensuring that they all arrive and are handled in the right order.

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

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

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

Status of This Memo

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

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

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

Copyright Notice

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

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

Table of Contents

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

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

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

- reducing implementation complexity in order to minimize code size,
- reducing message sizes in order to minimize the number of fragments needed for each message (in turn to maximize the probability of delivery of the message), the amount of transmission power needed and the loading of the limited-bandwidth channel,
- reducing requirements on the environment such as stable storage, good sources of randomness or user interaction capabilities.

CoAP is based on datagram transports such as UDP, which limit the maximum size of resource representations that can be transferred without creating unreasonable levels of IP fragmentation. In addition, not all resource representations will fit into a single link layer packet of a constrained network, which may cause adaptation layer fragmentation even if IP layer fragmentation is not required. Using fragmentation (either at the adaptation layer or at the IP layer) for the transport of larger representations would be possible up to the maximum size of the underlying datagram protocol (such as UDP), but the fragmentation/reassembly process burdens the lower layers with conversation state that is better managed in the application layer.
The present specification defines a pair of CoAP options to enable block-wise access to resource representations. The Block options provide a minimal way to transfer larger resource representations in a block-wise fashion. The overriding objective is to avoid the need for creating conversation state at the server for block-wise GET requests. (It is impossible to fully avoid creating conversation state for POST/PUT, if the creation/replacement of resources is to be atomic; where that property is not needed, there is no need to create server conversation state in this case, either.)

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

- Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- The transfer of each block is acknowledged, enabling individual retransmission if required.
- Both sides have a say in the block size that actually will be used.
- The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- If needed, the Block options can also be used (without changes) to provide random access to power-of-two sized blocks within a resource representation.

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

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

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

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

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

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

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

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

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

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

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

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

2.2. Structure of a Block Option

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

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

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

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

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

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

Figure 1: Block option value

The block size is encoded using a three-bit unsigned integer (0 for $2^{**4}$ to 6 for $2^{**10}$ bytes), which we call the "SZX" ("size exponent"); the actual block size is then $2^{**(SZX + 4)}$. SZX is transferred in the three least significant bits of the option value (i.e., "val & 7" where "val" is the value of the option).

The fourth least significant bit, the M or "more" bit ("val & 8"), indicates whether more blocks are following or the current block-wise transfer is the last block being transferred.

The option value divided by sixteen (the NUM field) is the sequence number of the block currently being transferred, starting from zero. The current transfer is therefore about the "size" bytes starting at byte "NUM << (SZX + 4)".

Implementation note: As an implementation convenience, "((val & ~0xF) << (val & 7))", i.e., the option value with the last 4 bits masked out, shifted to the left by the value of SZX, gives the byte position of the first byte of the block being transferred.

More specifically, within the option value of a Block1 or Block2 Option, the meaning of the option fields is defined as follows:
NUM: Block Number, indicating the block number being requested or provided. Block number 0 indicates the first block of a body (i.e., starting with the first byte of the body).

M: More Flag ("not last block"). For descriptive usage, this flag, if unset, indicates that the payload in this message is the last block in the body; when set it indicates that there are one or more additional blocks available. When a Block2 Option is used in a request to retrieve a specific block number ("control usage"), the M bit MUST be sent as zero and ignored on reception. (In a Block1 Option in a response, the M flag is used to indicate atomicity, see below.)

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

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

2.3. Block Options in Requests and Responses

The Block options are used in one of three roles:

- In descriptive usage, i.e., a Block2 Option in a response (such as a 2.05 response for GET), or a Block1 Option in a request (a PUT or POST):
  * The NUM field in the option value describes what block number is contained in the payload of this message.
  * The M bit indicates whether further blocks need to be transferred to complete the transfer of that body.
* The block size implied by SZX MUST match the size of the payload in bytes, if the M bit is set. (SZX does not govern the payload size if M is unset). For Block2, if the request suggested a larger value of SZX, the next request MUST move SZX down to the size given in the response. (The effect is that, if the server uses the smaller of (1) its preferred block size and (2) the block size requested, all blocks for a body use the same block size.)

- A Block2 Option in control usage in a request (e.g., GET):
  * The NUM field in the Block2 Option gives the block number of the payload that is being requested to be returned in the response.
  * In this case, the M bit has no function and MUST be set to zero.
  * The block size given (SZX) suggests a block size (in the case of block number 0) or repeats the block size of previous blocks received (in the case of a non-zero block number).

- A Block1 Option in control usage in a response (e.g., a 2.xx response for a PUT or POST request):
  * The NUM field of the Block1 Option indicates what block number is being acknowledged.
  * If the M bit was set in the request, the server can choose whether to act on each block separately, with no memory, or whether to handle the request for the entire body atomically, or any mix of the two.

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

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

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

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

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

2.4. Using the Block2 Option

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

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

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

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

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

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

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

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

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

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

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

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

2.6. Combining Block-wise Transfers with the Observe Option

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

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

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

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

See Section 3.4 for examples.

2.7. Combining Block1 and Block2

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

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

2.8. Combining Block2 with Multicast

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

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

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

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

2.9. Response Codes

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

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

2.9.2. 4.08 Request Entity Incomplete

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

2.9.3. 4.13 Request Entity Too Large

In [RFC7252], section 5.9.2.8, the response code 4.13 (Request Entity Too Large) is defined to be like HTTP 413 "Request Entity Too Large". [RFC7252] also recommends that this response SHOULD include a Size1 Option (Section 4) to indicate the maximum size of request entity the server is able and willing to handle, unless the server is not in a position to make this information available.

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

2.10. Caching Considerations

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

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

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

3. Examples

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

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

3.1. Block2 Examples

The first example (Figure 2) shows a GET request that is split into three blocks. The server proposes a block size of 128, and the client agrees. The first two ACKs contain 128 bytes of payload each, and third ACK contains between 1 and 128 bytes.
In the second example (Figure 3), the client anticipates the block-wise transfer (e.g., because of a size indication in the link-format description [RFC6690]) and sends a block size proposal. All ACK messages except for the last carry 64 bytes of payload; the last one carries between 1 and 64 bytes.

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

Figure 2: Simple block-wise GET

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

<table>
<thead>
<tr>
<th>CLIENT</th>
<th>SERVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON [MID=1234], GET, /status, 2:0/0/64</td>
<td></td>
</tr>
<tr>
<td>&lt;------- ACK [MID=1234], 2.05 Content, 2:0/1/64</td>
<td></td>
</tr>
<tr>
<td>CON [MID=1235], GET, /status, 2:1/0/64</td>
<td></td>
</tr>
<tr>
<td>&lt;------- ACK [MID=1235], 2.05 Content, 2:1/1/64</td>
<td></td>
</tr>
<tr>
<td>CON [MID=1238], GET, /status, 2:4/0/64</td>
<td></td>
</tr>
<tr>
<td>&lt;------- ACK [MID=1238], 2.05 Content, 2:4/1/64</td>
<td></td>
</tr>
<tr>
<td>CON [MID=1239], GET, /status, 2:5/0/64</td>
<td></td>
</tr>
<tr>
<td>&lt;------- ACK [MID=1239], 2.05 Content, 2:5/0/64</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Block-wise GET with early negotiation

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

![Figure 4: Block-wise GET with late negotiation](image)

In all these (and the following) cases, retransmissions are handled by the CoAP message exchange layer, so they don’t influence the block operations (Figure 5, Figure 6).
Figure 5: Block-wise GET with late negotiation and lost CON

Figure 6: Block-wise GET with late negotiation and lost ACK
3.2. Block1 Examples

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

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

Figure 7: Simple atomic block-wise PUT

A stateless server that simply builds/updates the resource in place (statelessly) may indicate this by not setting the more bit in the response (Figure 8); in this case, the response codes are valid separately for each block being updated. This is of course only an acceptable behavior of the server if the potential inconsistency present during the run of the message exchange sequence does not lead to problems, e.g. because the resource being created or changed is not yet or not currently in use.
Finally, a server receiving a block-wise PUT or POST may want to indicate a smaller block size preference (Figure 9). In this case, the client SHOULD continue with a smaller block size; if it does, it MUST adjust the block number to properly count in that smaller size.

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

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

```
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, 1:5/0/32   ------>
<------- 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.
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.
3.4. Combining Observe and Block2

In the following example, the server first sends a direct response (Observe sequence number 62350) to the initial GET request (the resulting block-wise transfer is as in Figure 4 and has therefore been left out). The second transfer is started by a 2.05 notification that contains just the first block (Observe sequence number 62354); the client then goes on to obtain the rest of the blocks.
2.05  |  Token: 0xfb
    |  Block2: 0/1/128
    |  Observe: 62350
    |  ETag: 6f00f38e
    |  Payload: [128 bytes]

(Usual GET transfer left out)

...(Notification of first block:)

<-----+     Header: 2.05 0x4145af9c
2.05 |  Token: 0xfb
    |  Block2: 0/1/128
    |  Observe: 62354
    |  ETag: 6f00f392
    |  Payload: [128 bytes]

|--|--->   Header: 0x6000af9c
     (Retrieval of remaining blocks)

++++++>  Header: GET 0x41011637
    |  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
    |  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 example 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 |
4. The Size2 and Size1 Options

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

This specification defines two CoAP Options, Size1 for indicating the size of the representation transferred in requests, and Size2 for indicating the size of the representation transferred in responses. (Size1 is already defined in [RFC7252] for the narrow case of indicating in 4.13 responses the maximum size of request payload that the server is able and willing to handle.)

The Size2 Option may be used for two purposes:

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

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

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

- in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation, measured in bytes ("size indication").
in a 4.13 response, to indicate the maximum size that would have been acceptable [RFC7252], measured in bytes.

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

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

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

Table 2: Size Option Numbers

Implementation Notes:

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

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

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

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

5. HTTP Mapping Considerations

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

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

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

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

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

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

Table 3: CoAP Option Numbers

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

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

Table 4: CoAP Response Codes

7. Security Considerations

Providing access to blocks within a resource may lead to surprising vulnerabilities. Where requests are not implemented atomically, an attacker may be able to exploit a race condition or confuse a server by inducing it to use a partially updated resource representation. Partial transfers may also make certain problematic data invisible to intrusion detection systems; it is RECOMMENDED that an intrusion detection system (IDS) that analyzes resource representations transferred by CoAP implement the Block options to gain access to entire resource representations. Still, approaches such as transferring even-numbered blocks on one path and odd-numbered blocks on another path, or even transferring blocks multiple times with different content and obtaining a different interpretation of temporal order at the IDS than at the server, may prevent an IDS from seeing the whole picture. These kinds of attacks are well understood from IP fragmentation and TCP segmentation; CoAP does not add fundamentally new considerations.
Where access to a resource is only granted to clients making use of specific security associations, all blocks of that resource MUST be subject to the same security checks; it MUST NOT be possible for unprotected exchanges to influence blocks of an otherwise protected resource. As a related consideration, where object security is employed, PUT/POST should be implemented in the atomic fashion, unless the object security operation is performed on each access and the creation of unusable resources can be tolerated.

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

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

7.1. Mitigating Resource Exhaustion Attacks

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

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

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

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

8. Acknowledgements

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

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

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

9. References

9.1. Normative References

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


9.2. Informative References


Authors’ Addresses

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

Phone: +49-421-218-63921
Email: cabo@tzi.org
Zach Shelby (editor)
ARM
150 Rose Orchard
San Jose, CA 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com
Guidelines for HTTP-CoAP Mapping Implementations
draft-ietf-core-http-mapping-07

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 4, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .......................................... 3
2. Terminology ............................................ 4
3. HTTP-CoAP Reverse Proxy ................................. 5
4. Use Cases ............................................... 6
5. URI Mapping ............................................ 7
   5.1. URI Terminology .................................... 8
   5.2. Default Mapping .................................... 8
      5.2.1. Optional Scheme Omission ........................ 8
      5.2.2. Encoding Caveats ................................ 9
   5.3. URI Mapping Template ............................... 9
      5.3.1. Simple Form .................................... 9
      5.3.2. Enhanced Form .................................. 11
   5.4. Discovery ........................................... 13
      5.4.1. Discovering CoAP Resources ..................... 13
      5.4.2. Examples ...................................... 14
6. Media Type Mapping ...................................... 15
   6.1. Overview .......................................... 15
   6.2. ‘application/coap-payload’ Media Type .............. 17
   6.3. Loose Media Type Mapping ......................... 17
   6.4. Media Type to Content Format Mapping Algorithm ... 18
   6.5. Content Transcoding ................................ 19
      6.5.1. General ........................................ 19
      6.5.2. CoRE Link Format .............................. 20
      6.5.3. Diagnostic Messages ............................ 20
7. Response Code Mapping .................................. 20
8. Additional Mapping Guidelines ......................... 23
   8.1. Caching and Congestion Control .................... 23
   8.2. Cache Refresh via Observe ........................ 23
   8.3. Use of CoAP Blockwise Transfer ................... 24
   8.4. Security Translation ................................ 25
   8.5. CoAP Multicast .................................... 25
   8.6. Timeouts .......................................... 26
   8.7. Miscellaneous ..................................... 26
9. IANA Considerations .................................... 26
   9.1. New ‘core.hc’ Resource Type ....................... 26
   9.2. New ‘coap-payload’ Internet Media Type ........... 27
1. Introduction

CoAP [RFC7252] has been designed with the twofold aim to be an application protocol specialized for constrained environments and to be easily used in REST architectures such as the Web. The latter goal has led to define CoAP to easily interoperate with HTTP [RFC7230] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [RFC7252] describes the fundamentals of the CoAP-to-HTTP and the HTTP-to-CoAP cross-protocol mapping process. However, implementing such a cross-protocol proxy can be complex, and many details regarding its internal procedures and design choices require further elaboration. Therefore, a first goal of this document is to provide more detailed information to proxy designers and implementers, to help build proxies that correctly inter-work with existing CoAP and HTTP implementations.

The second goal of this informational document is to define a consistent set of guidelines that a HTTP-to-CoAP proxy implementation MAY adhere to. The main reason for adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability. (For example, a proxy conforming to these guidelines made by vendor A can be easily replaced by a proxy from vendor B that also conforms to the guidelines.)

This document is organized as follows:

- Section 2 describes terminology to identify proxy types, mapping approaches and proxy deployments;
- Section 3 introduces the reverse HTTP-CoAP proxy;
- Section 4 lists use cases in which HTTP clients need to contact CoAP servers;
Section 5 introduces a default HTTP-to-CoAP URI mapping syntax;

Section 6 describes how to map HTTP media types to CoAP content formats and vice versa;

Section 7 describes how to map CoAP responses to HTTP responses;

Section 8 describes additional mapping guidelines related to caching, congestion, timeouts and CoAP blockwise [I-D.ietf-core-block] transfers;

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

2. Terminology

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

HC Proxy: a proxy performing a cross-protocol mapping, in the context of this document a HTTP-CoAP mapping. A Cross-Protocol Proxy can behave as a Forward Proxy, Reverse Proxy or Interception Proxy. In this document we focus on the Reverse Proxy case.

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

Reverse Proxy: as in [RFC7230], a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server’s protocol. A Reverse HC Proxy behaves as an origin (HTTP) server on its connection towards the (HTTP) client and as a (CoAP) client on its connection towards the (CoAP) origin server. The (HTTP) client uses the "origin-form" (Section 5.3.1 of [RFC7230]) as a request-target URI.

Interception Proxy [RFC3040]: a proxy that receives inbound traffic flows through the process of traffic redirection; transparent to the client.
Placement terms: a Server-Side proxy is placed in the same network domain as the server; conversely a Client-Side proxy is placed in the same network domain as the client. In any other case, the proxy is said to be External.

Note that a Reverse Proxy appears to a client as an origin server while a Forward Proxy does not, so, when communicating with a Reverse Proxy a client may be unaware it is communicating with a proxy at all.

3. HTTP-CoAP Reverse Proxy

A Reverse HTTP-CoAP Proxy (HC proxy) is accessed by clients only supporting HTTP, and handles their HTTP requests by mapping these to CoAP requests, which are forwarded to CoAP servers; mapping back received CoAP responses to HTTP responses. This mechanism is transparent to the client, which may assume that it is communicating with the intended target HTTP server. In other words, the client accesses the proxy as an origin server using the "origin-form" (Section 5.3.1 of [RFC7230]) as a request target.

See Figure 1 for an example deployment scenario. Here an HC Proxy is placed server-side, at the boundary of the Constrained Network domain, to avoid any HTTP traffic on the Constrained Network and to avoid any (unsecured) CoAP multicast traffic outside the Constrained Network. The DNS server is used by the HTTP Client to resolve the IP address of the HC Proxy and optionally also by the HC Proxy to resolve IP addresses of CoAP servers.
Other placement options for the HC Proxy (not shown) are client-side, which is in the same domain as the HTTP Client; or external, which is both outside the HTTP Client’s domain and the CoAP servers’ domain.

Normative requirements on the translation of HTTP requests to CoAP requests and of the CoAP responses back to HTTP responses are defined in Section 10.2 of [RFC7252]. However, that section only considers the case of a Forward HC Proxy in which a client explicitly indicates it targets a request to a CoAP server, and does not cover all aspects of proxy implementation in detail. This document provides guidelines and more details for the implementation of a Reverse HC Proxy, which MAY be followed in addition to the normative requirements. Note that most of the guidelines also apply to an Intercepting HC Proxy.

4. Use Cases

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

1. Smartphone and home sensor: A smartphone can access directly a CoAP home sensor using an authenticated ‘https’ request, if its home router contains an HC proxy. An HTML5 application on the smartphone can provide a friendly UI to the user using standard (HTTP) networking functions of HTML5.
2. Legacy building control application without CoAP: A building control application that uses HTTP but not CoAP, can check the status of CoAP sensors and/or actuators via an HC proxy.

3. Making sensor data available to 3rd parties: For demonstration or public interest purposes, a HC proxy may be configured to expose the contents of a CoAP sensor to the world via the web (HTTP and/or HTTPS). Some sensors might only handle secure 'coaps' requests, therefore the proxy is configured to translate any request to a 'coaps' secured request. The HC proxy is furthermore configured to only pass through GET requests in order to protect the constrained network. In this way even unattended HTTP clients, such as web crawlers, may index sensor data as regular web pages.

5. URI Mapping

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

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

URI Mapping is the process through which the URI of a CoAP resource is transformed into an HTTP URI so that:

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

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

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

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

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

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

5.2. Default Mapping

The default mapping is for the Target CoAP URI to be appended as-is to a base URI provided by the HC proxy, to form the Hosting HTTP URI.

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

Provided a correct Target CoAP URI, the Hosting HTTP URI resulting from the default mapping is always syntactically correct. Furthermore, the Target CoAP URI can always be extracted unambiguously from the Hosting HTTP URI. Also, it is worth noting that, using the default mapping, a query component in the target CoAP resource URI is naturally encoded into the query component of the Hosting URI, e.g.: coap://s.example.com/light?dim=5 becomes http://p.example.com/HC/coap://s.example.com/light?dim=5.

There is no default for the base URI. Therefore, it is either known in advance, e.g. as a configuration preset, or dynamically discovered using the mechanism described in Section 5.4.

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

5.2.1. Optional Scheme Omission

When found in a Hosting HTTP URI, the scheme (i.e., "coap" or "coaps"), the scheme component delimiter (":") and the double slash
("/"") preceding the authority MAY be omitted. In such case, a local default - not defined by this document - applies.

So, http://p.example.com/hc/s.coap.example.com/foo could either represent the target coap://s.coap.example.com/foo or coaps://s.coap.example.com/foo depending on application specific presets.

5.2.2. Encoding Caveats

When the authority of the Target CoAP URI is given as an IPv6address, then the surrounding square brackets MUST be percent-encoded in the Hosting HTTP URI, in order to comply with the syntax defined in Section 3.3. of [RFC3986] for a URI path segment. E.g.: coap://[2001:db8::1]/light?on becomes http://p.example.com/hc/coap://%5B2001:db8::1%5D/light?on.

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

5.3. URI Mapping Template

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

When instantiated, an URI Mapping Template is always concatenated to a base URI provided by the HC proxy via discovery (see Section 5.4), or by other means.

A simple form (Section 5.3.1) and an enhanced form (Section 5.3.2) are provided to fit different users’ requirements.

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

5.3.1. Simple Form

The simple form MUST be used for mappings where the Target CoAP URI is going to be copied (using rules of Section 5.2.2) at some fixed position into the Hosting HTTP URI.

The following template variables MUST be used in mutual exclusion in a template definition:
cu = coap-URI ; from [RFC7252], Section 6.1
su = coaps-URI ; from [RFC7252], Section 6.2
tu = cu / su

The same considerations as in Section 5.2.1 apply, in that the CoAP scheme may be omitted from the Hosting HTTP URI.

5.3.1.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use http://p.example.com/hc as the base URI. Note that these examples all define mapping templates that deviate from the default template of Section 5.2 to be able to illustrate the use of the above template variables.

1. "coap" URI is a query argument of the Hosting HTTP URI:

?coap_target_uri={+cu}

coap://s.example.com/light


2. "coaps" URI is a query argument of the Hosting HTTP URI:

?coaps_target_uri={+su}

coaps://s.example.com/light

http://p.example.com/hc?coaps_target_uri=coaps://s.example.com/light

3. Target CoAP URI as a query argument of the Hosting HTTP URI:
?target_uri={+tu}
coap://s.example.com/light
http://p.example.com/hc?target_uri=coap://s.example.com/light
or
coops://s.example.com/light
http://p.example.com/hc?target_uri=coaps://s.example.com/light

4. Target CoAP URI in the path component of the Hosting HTTP URI
(i.e., the default URI Mapping template):

/{+tu}
coap://s.example.com/light
http://p.example.com/hc/coap://s.example.com/light
or
coops://s.example.com/light
http://p.example.com/hc/coaps://s.example.com/light

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

?coap_uri={+cu}
coap://s.example.com/light
http://p.example.com/hc?coap_uri=s.example.com/light

5.3.2. Enhanced Form

The enhanced form can be used to express more sophisticated mappings,
i.e., those that do not fit into the simple form.
There MUST be at most one instance of each of the following template variables in a template definition:

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

5.3.2.1. Examples

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

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

{+s}{+hp}{+p}{+qq}
coap://s.example.com/light
http://p.example.com/hc/coap/s.example.com/light
or
coop://s.example.com/light?on
http://p.example.com/hc/coap/s.example.com/light?on

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

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

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

Along with it, the new target attribute "hct" is defined in this document. This attribute MAY be returned in a "core.hc" link to provide the URI Mapping Template associated to the mapping resource. The default template given in Section 5.2, i.e., {+tu}, MUST be assumed if no "hct" attribute is found in the returned link. If a "hct" attribute is present in the returned link, then a compliant client MUST use it to create the Hosting HTTP URI.

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

5.4.1. Discovering CoAP Resources

For a HTTP client, it may be unknown which CoAP resources are available through a HC Proxy. By default an HC Proxy does not support a method to discover all CoAP resources. However, if an HC Proxy is integrated with a Resource Directory ([I-D.ietf-core-resource-directory]) function, an HTTP client can
discover all CoAP resources of its interest by doing an RD Lookup to the HC Proxy, via HTTP. This is possible because a single RD can support both CoAP and HTTP interfaces simultaneously. Of course the HTTP client will this way only discover resources that have been previously registered onto this RD by CoAP devices.

5.4.2. Examples

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

  Req: GET coap://[ff02::1]/.well-known/core?rt=core.hc
  Res: 2.05 Content
       </hc>;anchor="http://p.example.com";rt="core.hc"

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

  Req: GET coap://[ff02::1]/.well-known/core?rt=core.hc
  Res: 2.05 Content
       </hc>;anchor="http://p.example.com";
       rt="core.hc";hct="?uri={+tu}" 

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

- using the ‘application/link-format’ content type:

  Req: GET /.well-known/core?rt=core.hc HTTP/1.1
       Host: p.example.com
  Res: HTTP/1.1 200 OK
       Content-Type: application/link-format
       Content-Length: 18
       
       </hc>;rt="core.hc"
o using the 'application/link-format+json' content type as defined in [I-D.bormann-core-links-json]:

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

Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format+json
      Content-Length: 31
      
      ["href":"/hc","rt":"core.hc"]

o using the Link header:

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

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

o An HC proxy may expose two different base URIs to differentiate between Target CoAP resources in the "coap" and "coaps" scheme:

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

Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format+json
      Content-Length: 111
      
      [
       {"href":"/hc/plaintext","rt":"core.hc","hct":{"cu"}},
       {"href":"/hc/secure","rt":"core.hc","hct":{"su"}}
      ]

6. Media Type Mapping

6.1. Overview

An HC proxy needs to translate HTTP media types (Section 3.1.1.1 of [RFC7231]) and content encodings (Section 3.1.2.2 of [RFC7231]) into CoAP content formats (Section 12.3 of [RFC7252]) and vice versa.
Media type translation can happen in GET, PUT or POST requests going from HTTP to CoAP, and in 2.xx (i.e., successful) responses going from CoAP to HTTP. Specifically, PUT and POST need to map both the Content-Type and Content-Encoding HTTP headers into a single CoAP Content-Format option, whereas GET needs to map Accept and Accept-Encoding HTTP headers into a single CoAP Accept option. To generate the HTTP response, the CoAP Content-Format option is mapped back to a suitable HTTP Content-Type and Content-Encoding combination.

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

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

While the CoAP to HTTP direction has always a well defined mapping (with the exception examined in Section 6.2), the HTTP to CoAP direction is more problematic because the source set, i.e., potentially 1000+ IANA registered media types, is much bigger than the destination set, i.e., the mere 6 values initially defined in Section 12.3 of [RFC7252].

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

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

On the other side, when the HC proxy is a general purpose application layer gateway, being too strict could significantly reduce the amount of traffic that it’d be able to successfully forward. In this case, the "loose" media type mapping detailed in Section 6.3 MAY be implemented.
The latter grants more evolution of the surrounding ecosystem, at the cost of allowing more attack surface. In fact, as a result of such strategy, payloads would be forwarded more liberally across the unconstrained/constrained network boundary of the communication path. Therefore, when applied, other forms of access control must be set in place to avoid unauthorized users to deplete or abuse systems and network resources.

6.2. ‘application/coap-payload’ Media Type

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

For example, the CoAP content format ‘60’ ("application/cbor") would be represented by "application/coap-payload;cf=60", would ‘60’ be an unknown content format to the HC Proxy.

A HTTP client MAY use the media type "application/coap-payload" as a means to send a specific content format to a CoAP server via an HC Proxy if the client has determined that the HC Proxy does not directly support the type mapping it needs. This case may happen when dealing for example with newly registered, yet to be registered, or experimental CoAP content formats.

6.3. Loose Media Type Mapping

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

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

To remedy this loss of flexibility, we introduce the concept of a "loose" media type mapping, where media types that are specializations of a more generic media type can be aliased to their super-class and then mapped (if possible) to one of the CoAP content formats. For example, "application/soap+xml" can be aliased to "application/xml", which has a known conversion to CoAP. In the context of this "loose" media type mapping, "application/octet-

stream" can be used as a fallback when no better alias is found for a specific media type.

Table 1 defines the default lookup table for the "loose" media type mapping. Given an input media type, the table returns its best generalized media type using the most specific match i.e. the table entries are compared to the input in top to bottom order until an entry matches.

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

Table 1: Media type generalization lookup table

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

6.4. Media Type to Content Format Mapping Algorithm

This section defines the algorithm used to map an HTTP Internet media type to its correspondent CoAP content format.

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

Note that the algorithm may have side effects on the associated representation (see also Section 6.5).

In the following:

- C-T, C-E, and C-F stand for the values of the Content-Type (or Accept) HTTP header, Content-Encoding (or Accept-Encoding) HTTP header, and Content-Format CoAP option respectively.
- If C-E is not given it is assumed to be "identity".
- MAP is the mandatory lookup table, GMAP is the optional generalized table.
INPUT: C-T and C-E
OUTPUT: C-F or Fail

1. if no C-T: return Fail
2. C-F = MAP[C-T, C-E]
3. if C-F is not None: return C-F
4. if C-E is not "identity":
   5. if C-E is supported (e.g. gzip):
      6. decode the representation accordingly
      7. set C-E to "identity"
   8. else:
      9. return Fail
10. repeat steps 2. and 3.
11. if C-T allows a non-lossy transformation into \\ one of the supported C-F:
12. transcode the representation accordingly
13. return C-F
14. if GMAP is defined:
15. C-F = GMAP[C-T]
16. if C-F is not None: return C-F
17. return Fail

Figure 2

6.5. Content Transcoding

6.5.1. General

Payload content transcoding (e.g. see steps 11-14 of Figure 2) is an OPTIONAL feature. Implementations supporting this feature should provide a flexible way to define the set of transcodings allowed.

As noted in Section 6.4, the process of mapping the media type can have side effects on the forwarded entity body. This may be caused by the removal or addition of a specific content encoding, or because the HC proxy decides to transcode the representation to a different (compatible) format. The latter proves useful when an optimized version of a specific format exists. For example an XML-encoded resource could be transcoded to Efficient XML Interchange (EXI) format, or a JSON-encoded resource into CBOR [RFC7049], effectively achieving compression without losing any information.

However, it should be noted that in certain cases, transcoding can lose information in a non-obvious manner. For example, encoding an XML document using schema-informed EXI encoding leads to a loss of information when the destination does not know the exact schema version used by the encoder, which means that whenever the HC proxy transcodes an application/XML to application/EXI in-band metadata...
could be lost. Therefore, the implementer should always carefully verify such lossy payload transformations before triggering the transcoding.

6.5.2. CoRE Link Format

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

1. The client may be interested to retrieve original (unaltered) CoAP payloads through the HC proxy, not modified versions.

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

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

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

6.5.3. Diagnostic Messages

CoAP responses may, in certain error cases, contain a diagnostic message in the payload explaining the error situation, as described in Section 5.5.2 of [RFC7252]. In this scenario, the CoAP response diagnostic payload MUST NOT be returned as the regular HTTP payload (message body). Instead, the CoAP diagnostic payload must be used as the HTTP reason-phrase of the HTTP status line, as defined in Section 3.1.2 of [RFC7230], without any alterations, except those needed to comply to the reason-phrase ABNF definition.

7. Response Code Mapping

Table 2 defines the HTTP response status codes to which each CoAP response code SHOULD be mapped. This table complies with the requirements in Section 10.2 of [RFC7252] and is intended to cover all possible cases. Multiple appearances of a HTTP status code in the second column indicates multiple equivalent HTTP responses are
possible based on the same CoAP response code, depending on the conditions cited in the Notes (third column and text below table).

<table>
<thead>
<tr>
<th>CoAP Response Code</th>
<th>HTTP Status Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.01 Created</td>
<td>201 Created</td>
<td>1</td>
</tr>
<tr>
<td>2.02 Deleted</td>
<td>200 OK</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>204 No Content</td>
<td>2</td>
</tr>
<tr>
<td>2.03 Valid</td>
<td>304 Not Modified</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>200 OK</td>
<td>4</td>
</tr>
<tr>
<td>2.04 Changed</td>
<td>200 OK</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>204 No Content</td>
<td>2</td>
</tr>
<tr>
<td>2.05 Content</td>
<td>200 OK</td>
<td></td>
</tr>
<tr>
<td>4.00 Bad Request</td>
<td>400 Bad Request</td>
<td></td>
</tr>
<tr>
<td>4.01 Unauthorized</td>
<td>401 Unauthorized</td>
<td>5</td>
</tr>
<tr>
<td>4.02 Bad Option</td>
<td>400 Bad Request</td>
<td>6</td>
</tr>
<tr>
<td>4.03 Forbidden</td>
<td>403 Forbidden</td>
<td></td>
</tr>
<tr>
<td>4.04 Not Found</td>
<td>404 Not Found</td>
<td></td>
</tr>
<tr>
<td>4.05 Method Not Allowed</td>
<td>400 Bad Request</td>
<td>7</td>
</tr>
<tr>
<td>4.06 Not Acceptable</td>
<td>406 Not Acceptable</td>
<td></td>
</tr>
<tr>
<td>4.12 Precondition Failed</td>
<td>412 Precondition Failed</td>
<td></td>
</tr>
<tr>
<td>4.13 Request Ent. Too Large</td>
<td>413 Request Repr. Too Large</td>
<td></td>
</tr>
<tr>
<td>4.15 Unsupported Media Type</td>
<td>415 Unsupported Media Type</td>
<td></td>
</tr>
<tr>
<td>5.00 Internal Server Error</td>
<td>500 Internal Server Error</td>
<td></td>
</tr>
<tr>
<td>5.01 Not Implemented</td>
<td>501 Not Implemented</td>
<td></td>
</tr>
<tr>
<td>5.02 Bad Gateway</td>
<td>502 Bad Gateway</td>
<td></td>
</tr>
<tr>
<td>5.03 Service Unavailable</td>
<td>503 Service Unavailable</td>
<td>8</td>
</tr>
<tr>
<td>5.04 Gateway Timeout</td>
<td>504 Gateway Timeout</td>
<td></td>
</tr>
<tr>
<td>5.05 Proxying Not Supported</td>
<td>502 Bad Gateway</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 2: CoAP-HTTP Response Code Mappings

Notes:

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

2. The HTTP code is 200 or 204 respectively for the case that a CoAP server returns a payload or not. [RFC7231] Section 5.3 requires code 200 in case a representation of the action result is returned for DELETE/POST/PUT, and code 204 if not. Hence, a proxy SHOULD transfer any CoAP payload contained in a CoAP 2.02 response to the HTTP client using a 200 OK response.
3. HTTP code 304 (Not Modified) is sent if the HTTP client performed a conditional HTTP request and the CoAP server responded with 2.03 (Valid) to the corresponding CoAP validation request. Note that Section 4.1 of [RFC7232] puts some requirements on header fields that must be present in the HTTP 304 response.

4. A 200 response to a CoAP 2.03 occurs only when the HC proxy, for efficiency reasons, is caching resources and translated a HTTP request (without conditional request) to a CoAP request that includes ETag validation. The proxy receiving 2.03 updates the freshness of its cached representation and returns the entire representation to the HTTP client.

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

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

7. A CoAP 4.05 (Method Not Allowed) response SHOULD normally be mapped to a HTTP 400 (Method Not Allowed) code, because the HTTP 405 response would require specifying the supported methods – which are generally unknown. In this case the HC Proxy SHOULD also return a HTTP reason-phrase in the HTTP status line that starts with the string "405" in order to facilitate troubleshooting. However, if the HC proxy has more granular information about the supported methods for the requested resource (e.g. via a Resource Directory ([I-D.ietf-core-resource-directory])) then it MAY send back a HTTP 405 (Method Not Allowed) with a properly filled in "Allow" response-header field (Section 7.4.1 of [RFC7231]).

8. The value of the HTTP "Retry-After" response-header field is taken from the value of the CoAP Max-Age Option, if present.
9. This CoAP response can only happen if the proxy itself is configured to use a CoAP forward-proxy (Section 5.7 of [RFC7252]) to execute some, or all, of its CoAP requests.

8. Additional Mapping Guidelines

8.1. Caching and Congestion Control

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

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

If the HTTP client times out and drops the HTTP session to the HC proxy (closing the TCP connection) after the HTTP request was made, an HC proxy SHOULD wait for the associated CoAP response and cache it if possible. Subsequent requests to the HC proxy for the same resource can use the result present in cache, or, if a response has still to come, the HTTP requests will wait on the open CoAP request.

According to [RFC7252], a proxy MUST limit the number of outstanding interactions to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the HC proxy SHOULD also pose a limit to the number of concurrent CoAP requests pending on the same constrained network; further incoming requests MAY either be queued or dropped (returning 503 Service Unavailable). This limit and the proxy queueing/dropping behavior SHOULD be configurable. In order to effectively apply above congestion control, the HC proxy should be server-side placed.

Resources experiencing a high access rate coupled with high volatility MAY be observed [I-D.ietf-core-observe] by the HC proxy to keep their cached representation fresh while minimizing the number of CoAP traffic in the constrained network. See Section 8.2.

8.2. Cache Refresh via Observe

There are cases where using the CoAP observe protocol [I-D.ietf-core-observe] to handle proxy cache refresh is preferable to the validation mechanism based on ETag as defined in [RFC7252]. Such scenarios include, but are not limited to, sleepy CoAP nodes -- with possibly high variance in requests’ distribution -- which would greatly benefit from a server driven cache update mechanism. Ideal candidates for CoAP observe are also crowded or very low throughput
networks, where reduction of the total number of exchanged messages is an important requirement.

This subsection aims at providing a practical evaluation method to decide whether refreshing a cached resource R is more efficiently handled via ETag validation or by establishing an observation on R.

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

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

8.3. Use of CoAP Blockwise Transfer

An HC proxy SHOULD support CoAP blockwise transfers [I-D.ietf-core-block] to allow transport of large CoAP payloads while avoiding excessive link-layer fragmentation in constrained networks, and to cope with small datagram buffers in CoAP end-points as described in [RFC7252] Section 4.6.

An HC proxy SHOULD attempt to retry a payload-carrying CoAP PUT or POST request with blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. An HC proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than BLOCKWISE_THRESHOLD bytes. The value of BLOCKWISE_THRESHOLD is implementation-specific, for example it can be:

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

The value BLOCKWISE_THRESHOLD, or the parameters from which it is calculated, should be configurable in a proxy implementation. The maximum block size the proxy will attempt to use in CoAP requests should also be configurable.
The HC proxy SHOULD detect CoAP end-points not supporting blockwise transfers by checking for a 4.02 (Bad Option) response returned by an end-point in response to a CoAP request with a Block* Option, and subsequent absence of the 4.02 in response to the same request without Block* Options. This allows the HC proxy to be more efficient, not attempting repeated blockwise transfers to CoAP servers that do not support it. However, if a request payload is too large to be sent as a single CoAP request and blockwise transfer would be unavoidable, the proxy still SHOULD attempt blockwise transfer on such an end-point before returning the response 413 (Request Entity Too Large) to the HTTP client.

For improved latency an HC proxy MAY initiate a blockwise CoAP request triggered by an incoming HTTP request even when the HTTP request message has not yet been fully received, but enough data has been received to send one or more data blocks to a CoAP server already. This is particularly useful on slow client-to-proxy connections.

8.4. Security Translation

For the guidelines on security context translations for an HC proxy, see Section 10.2. A translation may involve e.g. applying a rule that any "https" request is translated to a "coaps" request, or e.g. applying a rule that a "https" request is translated to an unsecured "coap" request.

8.5. CoAP Multicast

An HC proxy MAY support CoAP multicast. If it does, the HC proxy sends out a multicast CoAP request if the Target CoAP URI’s authority is a multicast IP literal or resolves to a multicast IP address; assuming the proper security measures are in place to mitigate security risks of CoAP multicast (Section 10). If the security policies do not allow the specific CoAP multicast request to be made, the HC proxy SHOULD respond 403 (Forbidden).

If an HC proxy does not support CoAP multicast, it SHOULD respond 403 (Forbidden) to any valid HTTP request that maps to a CoAP multicast request.

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

8.6. Timeouts

When facing long delays of a CoAP server in responding, the HTTP client or any other proxy in between MAY timeout. Further discussion of timeouts in HTTP is available in Section 6.2.4 of [RFC7230].

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

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

where MAX\_RTT is defined in [RFC7252] and MAX\_SERVER\_RESPONSE\_DELAY is defined in [RFC7390]. An exception to this rule occurs when the HC proxy is configured with a HTTP response timeout value that is lower than above value T; then the lower value should be also used as the CoAP request timeout.

8.7. Miscellaneous

In certain use cases, constrained CoAP nodes do not make use of the DNS protocol. However even when the DNS protocol is not used in a constrained network, defining valid FQDN (i.e., DNS entries) for constrained CoAP servers, where possible, may help HTTP clients to access the resources offered by these servers via an HC proxy.

HTTP connection pipelining (section 6.3.2 of [RFC7230]) may be supported by an HC proxy. This is transparent to the CoAP servers: the HC proxy will serve the pipelined requests by issuing different CoAP requests. The HC proxy in this case needs to respect the NSTART limit of Section 4.7 of [RFC7252].

9. IANA Considerations

9.1. New ‘core.hc’ Resource Type

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

Attribute Value: core.hc

Description: HTTP to CoAP mapping base resource.
Reference: See Section 5.4.

9.2. New ‘coap-payload’ Internet Media Type

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

Type name: application

Subtype name: coap-payload

Required parameters:

cf - CoAP Content-Format integer in range 0-65535 denoting the content format of the CoAP payload carried.

Optional parameters: None

Encoding considerations:

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

Security considerations:

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

Interoperability considerations:

Published specification: (this I-D - TBD)

Applications that use this media type:

HTTP-to-CoAP Proxies.

Fragment identifier considerations: N/A

Additional information:

   Deprecated alias names for this type: N/A

   Magic number(s): N/A
10. Security Considerations

The security concerns raised in Section 9.2 of [RFC7230] also apply to the HC proxy scenario. In fact, the HC proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the HC proxy cannot be dropped, because the protocol translation function is the core duty of the HC proxy: it is a necessarily trusted, impossible to bypass, component in the communication path.

A reverse proxy deployed at the boundary of a constrained network is an easy single point of failure for reducing availability. As such, special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it has fewer limitations than the constrained devices it is serving.

The following sub paragraphs categorize and discuss a set of specific security issues related to the translation, caching and forwarding functionality exposed by an HC proxy.
10.1. Traffic Overflow

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

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

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

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

10.2. Handling Secured Exchanges

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

An HC proxy SHOULD implement explicit rules for security context translations. A translation may involve e.g. applying a rule that any "https" unicast request is translated to a "coaps" request, or e.g. applying a rule that a "https" request is translated to an unsecured "coap" request. Another rule could specify the security policy and parameters used for DTLS connections. Such rules will largely depend on the application and network context in which a proxy operates. These rules SHOULD be configurable in an HC proxy.

If a policy for access to ‘coaps’ URIs is configurable in an HC proxy, it is RECOMMENDED that the policy is by default configured to disallow access to any ‘coaps’ URI by a HTTP client using an unsecured (non-TLS) connection. Naturally, a user MAY reconfigure the policy to allow such access in specific cases.
By default, an HC proxy SHOULD reject any secured client request if there is no configured security policy mapping. This recommendation MAY be relaxed in case the destination network is believed to be secured by other, complementary, means. E.g.: assumed that CoAP nodes are isolated behind a firewall (e.g. as in the SS HC proxy deployment shown in Figure 1), the HC proxy may be configured to translate the incoming HTTPS request using plain CoAP (NoSec mode).

The HTTP-CoAP URI mapping (defined in Section 5) MUST NOT map to HTTP a CoAP resource intended to be only accessed securely.

A secured connection that is terminated at the HC proxy, i.e., the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The HC proxy SHOULD NOT cache any secured content to avoid any leak of secured information. However, in some specific scenario, a security/efficiency trade-off could motivate caching secured information; in that case the caching behavior MAY be tuned to some extent on a per-resource basis.

10.3. Proxy and CoAP Server Resource Exhaustion

If the HC proxy implements the low-latency optimization of Section 8.3 intended for slow client-to-proxy connections, the Proxy may become vulnerable to a resource exhaustion attack. In this case an attacking client could initiate multiple requests using a relatively large message body which is (after an initial fast transfer) transferred very slowly to the Proxy. This would trigger the HC proxy to create state for a blockwise CoAP request per HTTP request, waiting for the arrival of more data over the HTTP/TCP connection. Such attacks can be mitigated in the usual ways for HTTP servers using for example a connection time limit along with a limit on the number of open TCP connections per IP address.

10.4. URI Mapping

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

DoS attack on the constrained/CoAP network.
To mitigate, by default deny any Target CoAP URI whose authority is (or maps to) a multicast address. Then explicitly white-list multicast resources/authorities that are allowed to be de-referenced. See also Section 8.5.

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

Reduced privacy due to the mechanics of the URI mapping. The internal CoAP Target resource is totally transparent from outside. An HC proxy can mitigate by implementing a HTTPS-only interface, making the Target CoAP URI totally opaque to a passive attacker.

11. Acknowledgements

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

Thanks to Carsten Bormann, Zach Shelby, Michele Rossi, Nicola Bui, Michele Zorzi, Klaus Hartke, Cullen Jennings, Kepeng Li, Brian Frank, Peter Saint-Andre, Kerry Lynn, Linyi Tian, Dorothy Gellert, Francesco Corazza for helpful comments and discussions that have shaped the document.

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

12. References

12.1. Normative References

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

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


12.2. Informative References


Appendix A. Change Log

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

Changes from ietf-06 to ietf-07:

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

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

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

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

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

Changes from ietf-05 to ietf-06:

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

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

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

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

- Addressed Ticket #376 (CoAP 4.05 response can’t be translated to HTTP 405 by HC proxy) in Section 7 by use of empty ‘Allow’ header.
- Removed details on the pros and cons of HC proxy placement options;
- Addressed review comments of Carsten Bormann;
- Clarified failure in mapping of HTTP Accept headers (Section 6.3);
- Clarified detection of CoAP servers not supporting blockwise (Section 8.3);
- Changed CoAP request timeout min value to MAX_RTT + MAX_SERVER_RESPONSE_DELAY (Section 8.6);
- Added security section item (Section 10.3) related to use of CoAP blockwise transfers;
- Many editorial improvements.

Changes from ietf-04 to ietf-05:

- Addressed Ticket #366 (Mapping of CoRE Link Format payloads to be valid in HTTP Domain?) in Section 6.3.3.2 (Content Transcoding - CORE Link Format);
- Addressed Ticket #375 (Add requirement on mapping of CoAP diagnostic payload) in Section 6.3.3.3 (Content Transcoding - Diagnostic Messages);
- Addressed comment from Yusuke (http://www.ietf.org/mail-archive/web/core/current/msg05491.html) in Section 6.3.3.1 (Content Transcoding - General);
- Various editorial improvements.

Changes from ietf-03 to ietf-04:

- Expanded use case descriptions in Section 4;
- Fixed/enhanced discovery examples in Section 5.4.1;
- Addressed Ticket #365 (Add text on media type conversion by HTTP-CoAP proxy) in new Section 6.3.1 (Generalized media type mapping) and new Section 6.3.2 (Content translation);
- Updated HTTPBis WG draft references to recently published RFC numbers.
- Various editorial improvements.
Changes from ietf-02 to ietf-03:

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

Changes from ietf-01 to ietf-02:

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

Changes from ietf-00 to ietf-01:

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

Authors’ Addresses

Angelo P. Castellani  
University of Padova  
Via Gradenigo 6/B  
Padova 35131  
Italy  
Email: angelo@castellani.net

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland  
Email: salvatore.loreto@ericsson.com
CoRE Interfaces
draft-ietf-core-interfaces-03

Abstract

This document defines well-known REST interface descriptions for Batch, Sensor, Parameter and Actuator types for use in constrained web servers using the CoRE Link Format. A short reference is provided for each type that can be efficiently included in the interface description attribute of the CoRE Link Format. These descriptions are intended to be for general use in resource designs or for inclusion in more specific interface profiles. In addition, this document defines the concepts of Function Set and Binding. The former is the basis element to create RESTful profiles and the latter helps the configuration of links between resources located on one or more endpoints.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

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

The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g., 8-bit microcontrollers with limited RAM and ROM) and networks (e.g., 6LoWPAN). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690] and can be used by CoAP [RFC7252] or HTTP servers. The CoRE Link Format defines an attribute that can be used to describe the REST interface of a resource, and may include a link to a description document. This memo describes how other specifications can combine resources with a well-known interface to create new CoRE RESTful profiles. A CoRE profile is based on the concept of Function Set, which is a group of REST resources providing a service in a distributed system. In addition, the notion of Binding is introduced in order to create a synchronization link between two resources. This document also defines well-known interface descriptions for Batch, Sensor, Parameter and Actuator types to compose new Function Sets or for standalone use in a constrained web server. A short reference is provided for each type that can be efficiently included in the interface description (if=) attribute of the CoRE Link Format.

2. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. This specification makes use of the following additional terminology:

Function Set: A group of well-known REST resources that provides a particular service.
Profile: A group of well-known Function Sets defined by a specification.

Device: An IP smart object running a web server that hosts a group of Function Set instances from a profile.

Service Discovery: The process making it possible for a web client to automatically detect devices and Function Sets offered by these devices on a CoRE network.

Resource Discovery: The process allowing a web client to identify resources being hosted on a web server.

Gradual Reveal: A REST design where resources are discovered progressively using Web Linking.

Binding: A unidirectional logical link between a source resource and a destination resource.

3. Function Set

This section defines how a set of REST resources can be created called a function set. A Function Set is similar to a function block in the sense that it consists of input, output and parameter resources and contains internal logic. A Function Set can have a subset of mandatory inputs, outputs and parameters to provide minimum interoperability. It can also be extended with manufacturer/user-specific resources. A device is composed of one or more Function Set instances.

An example of function sets can be found from the CoRE Resource Directory specification that defines REST interfaces for registration, group and lookup [I-D.ietf-core-resource-directory]. The OMA Lightweight M2M standard [REF] also defines a function set structure called an Objects that use integer path, instance and resource URI segments. OMA Objects can be defined and then registered with an OMA maintained registry [REF]. This section is simply meant as a guideline for the definition of other such REST interfaces, either custom or part of other specifications.

3.1. Defining a Function Set

In a Function Set, types of resources are defined. Each type includes a human readable name, a path template, a Resource Type for discovery, the Interface Definition and the data type and allowed values. A Function Set definition may also include a field indicating if a sub-resource is mandatory or optional.
3.1.1. Path template

A Function Set is a container resource under which its sub-resources are organized. The profile defines the path to each resource of a Function Set in a path template. The template can contain either relative paths or absolute paths depending on the profile needs. An absolute Function Set should be located at its recommended root path on a web server, however it can be located under an alternative path if necessary (for example multi-purpose devices, gateways etc.). A relative Function Set can be instantiated as many times as needed on a web server with an arbitrary root path. However some Function Sets (e.g. device description) only make sense as singletons.

The path template includes a possible index {#} parameter, and possible fixed path segments. The index {#} allows for multiple instances of this type of resource, and can be any string. The root path and the indexes are the only variable elements in a path template. All other path segments should be fixed.

3.1.2. Resource Type

Each root resource of a Function Set is assigned a Resource Type parameter, therefore making it possible to discover it. Each sub-resource of a Function Set is also assigned a Resource Type parameter. This Resource Type is used for resource discovery and is usually necessary to discover optional resources supported on a specific device. The Resource Type of a Function Set may also be used for service discovery and can be exported to DNS-SD [RFC6763] for example.

The Resource Type parameter defines the value that should be included in the rt= field of the CoRE Link Format when describing a link to this resource. The value SHOULD be in the form "namespace.type" for root resources and "namespace.type.subtype" for sub-resources. This naming convention facilitates resource type filtering with the /.well-known/core resource. However a profile could allow mixing in foreign namespace references within a Function Set to import external references from other object models (e.g. SenML and UCUM).

3.1.3. Interface Description

The Interface Description parameter defines the REST interface for that type of resource. Several base interfaces are defined in Section 5 of this document. For a given profile, the Interface Description may be inferred from the Resource Type. In that case the Interface Description MAY be elided from link descriptions of resource types defined in the profile, but should be included for custom extensions to the profile.
The root resource of a Function Set should provide a list of links to its sub-resources in order to offer gradual reveal of resources. The CoRE Link List interface defined in Section 5.1 offers this functionality so a root resource should support this interface or a derived interface like CoRE Batch (See Section 5.2).

3.1.4. Data type

The Data Type field defines the type of value (and possible range) that is returned in response to a GET for that resource or accepted with a PUT. The interfaces defined in Section 5 make use of plain text and SenML Media types for the actual format of this data. A profile may restrict the list of supported content types for the CoRE interfaces or define new interfaces with new content types.

3.2. Discovery

A device conforming to a profile SHOULD make its resources discoverable by providing links to the resources on the path /.well-known/core as defined in [RFC6690]. All resources hosted on a device SHOULD be discoverable either with a direct link in /.well-known/core or by following successive links starting from /.well-known/core.

The root path of a Function Set instance SHOULD be directly referenced in /.well-known/core in order to offer discovery at the first discovery stage. A device with more than 10 individual resources SHOULD only expose Function Set instances in /.well-known/core to limit the size of this resource.

In addition, a device MAY register its resources to a Resource Directory using the registration interface defined in [I-D.ietf-core-resource-directory] if such a directory is available.

3.3. Versioning

A profile should track Function Set changes to avoid incompatibility issues. Evolutions in a Function Set SHOULD be backward compatible.

4. Bindings

In a M2M RESTful environment, endpoints exchange the content of their resources to operate the distributed system. Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human intervention and a so-called commissioning tool. In this document the abstract relationship between two resources is called a Binding.
The configuration phase necessitates the exchange of binding information so a format recognized by all CoRE endpoints is essential. This document defines a format based on the CoRE Link-Format to represent binding information along with the rules to define a binding method which is a specialized relationship between two resources. The purpose of a binding is to synchronize the content between a source resource and a destination resource. The destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address). Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method. The following table gives a summary of the binding methods described in more detail in Section 4.2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Identifier</th>
<th>Location</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polling</td>
<td>poll</td>
<td>Destination</td>
<td>GET</td>
</tr>
<tr>
<td>Observe</td>
<td>obs</td>
<td>Destination</td>
<td>GET + Observe</td>
</tr>
<tr>
<td>Push</td>
<td>push</td>
<td>Source</td>
<td>PUT</td>
</tr>
</tbody>
</table>

4.1. Format

Since Binding lies in the creation of a link between two resources, Web Linking and the CoRE Link-Format are a natural way to represent binding information. This involves the creation of a new relation type, purposely named "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource. The Web link attributes allow a fine-grained control of the type of synchronization exchange along with the conditions that trigger an update. This specification defines the attributes below:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binding method</td>
<td>bind</td>
<td>xsd:string</td>
</tr>
<tr>
<td>Minimum Period (s)</td>
<td>pmin</td>
<td>xsd:integer (&gt;0)</td>
</tr>
<tr>
<td>Maximum Period (s)</td>
<td>pmax</td>
<td>xsd:integer (&gt;0)</td>
</tr>
<tr>
<td>Change Step</td>
<td>st</td>
<td>xsd:decimal (&gt;0)</td>
</tr>
<tr>
<td>Greater Than</td>
<td>gt</td>
<td>xsd:decimal</td>
</tr>
<tr>
<td>Less Than</td>
<td>lt</td>
<td>xsd:decimal</td>
</tr>
</tbody>
</table>
Bind Method: This is the identifier of a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

Minimum Period: When present, the minimum period indicates the minimum time to wait (in seconds) before sending a new synchronization message (even if it has changed). In the absence of this parameter, the minimum period is up to the notifier.

Maximum Period: When present, the maximum period indicates the maximum time in seconds between two consecutive synchronization messages (regardless if it has changed). In the absence of this parameter, the maximum period is up to the notifier. The maximum period MUST be greater than the minimum period parameter (if present).

Change Step: When present, the change step indicates how much the value of a resource SHOULD change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification SHOULD be between pmin and pmax.

Greater Than: When present, Greater Than indicates the upper limit value the resource value SHOULD cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Greater Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

Less Than: When present, Less Than indicates the lower limit value the resource value SHOULD cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Less Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

4.2. Binding methods

A binding method defines the rules to generate the web-transfer exchanges that will effectively send content from the source resource to the destination resource. The description of a binding method must define the following aspects:

Identifier: This is the value of the "bind" attribute used to identify the method.
Location: This information indicates whether the binding entry is stored on the source or on the destination endpoint.

REST Method: This is the REST method used in the Request/Response exchanges.

Conditions: A binding method definition must state how the condition attributes of the abstract binding definition are actually used in this specialized binding.

This specification supports 3 binding methods described below.

Polling: The Polling method consists of sending periodic GET requests from the destination endpoint to the source resource and copying the content to the destination resource. The binding entry for this method MUST be stored on the destination endpoint. The destination endpoint MUST ensure that the polling frequency does not exceed the limits defined by the pmin and pmax attributes of the binding entry. The copying process MAY filter out content from the GET requests using value-based conditions (e.g., Change Step, Less Than, Greater Than).

Observe: The Observe method relies on the Publish/Subscribe pattern thus an observation relationship is created between the destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the CoAP Observation mechanism [I-D.ietf-core-observe] hence this method is only permitted when the resources are made available over CoAP. The binding entry for this method MUST be stored on the destination endpoint. The binding conditions are mapped as query string parameters (see Section 5.9).

Push: When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the binding condition attributes are satisfied for the source resource. The source endpoint MUST only send a notification request if the binding conditions are met. The binding entry for this method MUST be stored on the source endpoint.

4.3. Binding table

The binding table is a special resource that gives access to the bindings on a endpoint. A binding table resource MUST support the Binding interface defined in Section 5.8. A profile SHOULD allow only one resource table per endpoint.
5. Interface Descriptions

This section defines REST interfaces for Link List, Batch, Sensor, Parameter, Actuator and Binding table resources. Variants such as Linked Batch or Read-Only Parameter are also presented. Each type is described along with its Interface Description attribute value and valid methods. These are defined for each interface in the table below. These interfaces can support plain text and/or SenML Media types.

The if= column defines the Interface Description (if=) attribute value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this specification. Although these interface descriptions are intended to be used with the CoRE Link Format, they are applicable for use in any REST interface definition.

The Methods column defines the methods supported by that interface, which are described in more detail below.

<table>
<thead>
<tr>
<th>Interface</th>
<th>if=</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link List</td>
<td>core.ll</td>
<td>GET</td>
</tr>
<tr>
<td>Batch</td>
<td>core.b</td>
<td>GET, PUT, POST (where applicable)</td>
</tr>
<tr>
<td>Linked Batch</td>
<td>core.lb</td>
<td>GET, PUT, POST, DELETE (where applicable)</td>
</tr>
<tr>
<td>Sensor</td>
<td>core.s</td>
<td>GET</td>
</tr>
<tr>
<td>Parameter</td>
<td>core.p</td>
<td>GET, PUT</td>
</tr>
<tr>
<td>Read-only Parameter</td>
<td>core.rp</td>
<td>GET</td>
</tr>
<tr>
<td>Parameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actuator</td>
<td>core.a</td>
<td>GET, PUT, POST</td>
</tr>
<tr>
<td>Binding</td>
<td>core.bnd</td>
<td>GET, POST, DELETE</td>
</tr>
</tbody>
</table>

The following is an example of links in the CoRE Link Format using these interface descriptions. The resource hierarchy is based on a simple profile defined in Appendix A. These links are used in the subsequent examples below.
5.1. Link List

The Link List interface is used to retrieve (GET) a list of resources on a web server. The GET request SHOULD contain an Accept option with the application/link-format content type, however if the resource does not support any other form of GET methods the Accept option MAY be elided. The Accept option SHOULD only include the application/link-format content type. The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on a web server.

Link List is the base interface to provide gradual reveal of resources on a CoRE web server, hence the root resource of a Function Set SHOULD implement this interface or an extension of this interface.

The following example interacts with a Link List /d containing Parameter sub-resources /d/name, /d/model.

Req: GET /d (Accept:application/link-format)
Res: 2.05 Content (application/link-format)
</d/name>;rt="simple.dev.n";if="core.p",
</d/model>;rt="simple.dev.mdl";if="core.rp"

5.2. Batch

The Batch interface is used to manipulate a collection of sub-resources at the same time. The Batch interface type supports the same methods as its sub-resources, and can be used to read (GET), set (PUT) or toggle (POST) the values of those sub-resource with a single resource representation. The sub-resources of a Batch MAY be heterogeneous, a method used on the Batch only applies to sub-
resources that support it. For example Sensor interfaces do not support PUT, and thus a PUT request to a Sensor member of that Batch would be ignored. A batch requires the use of SenML Media types in order to support multiple sub-resources.

In addition, The Batch interface is an extension of the Link List interface and in consequence MUST support the same methods.

The following example interacts with a Batch /s with Sensor sub-resources /s/light, /s/temp and /s/humidity.

Req: GET /s
Res: 2.05 Content (application/senml+json)
{"e": [ 
  { "n": "light", "v": 123, "u": "lx" },
  { "n": "temp", "v": 27.2, "u": "degC" },
  { "n": "humidity", "v": 80, "u": "%RH" }]
}

5.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the web server, a Linked Batch is dynamically controlled by a web client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC5988] and the CoRE Link Format [RFC6690]. A request with a POST method and a content type of application/link-format simply appends new resources to the collection. The links in the payload MUST reference a resource on the web server with an absolute path. A DELETE request empties the current collection of links. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /l and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.
Req: POST /l (Content-type: application/link-format)  
</s/light>,</s/temp>  
Res: 2.04 Changed  

Req: GET /l  
Res: 2.05 Content (application/senml+json)  
{"e":[  
{ "n": "/s/light", "v": 123, "u": "lx" },  
{ "n": "/s/temp", "v": 27.2, "u": "degC" },  
]  
}

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

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

Req: GET /l  
Res: 2.05 Content (application/senml+json)  
{"e":[  
{ "n": "/s/light", "v": 123, "u": "lx" },  
{ "n": "/s/temp", "v": 27.2, "u": "degC" },  
{ "n": "/s/humidity", "v": 80, "u": "%RH" }],  
}

Req: DELETE /l  
Res: 2.04 Changed  

5.4. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.
5.5. Parameter

The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or set (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and setting a parameter.

Req: GET /d/name
Res: 2.05 Content (text/plain)
node5

Req: PUT /d/name (text/plain)
outdoor
Res: 2.04 Changed

5.6. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not set. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
5.7. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or a new actuator value set (PUT). In addition, this interface defines the use of POST (with no body) to toggle an actuator between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to set.

The following example shows requests for reading, setting and toggling an actuator (turning on a led).

Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0

Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed

Req: POST /a/1/led (text/plain)
Res: 2.04 Changed

Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0

5.8. Binding

The Binding interface is used to manipulate a binding table. A request with a POST method and a content type of application/link-format simply appends new bindings to the table. All links in the payload MUST have a relation type "boundTo". A GET request simply returns the current state of a binding table whereas a DELETE request empties the table.

The following example shows requests for adding, retrieving and deleting bindings in a binding table.
Internet-Draft                CoRE Interfaces                   July 2015

Req: POST /bnd (Content-type: application/link-format)
    <coap://sensor.example.com/s/light>;
        rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
Res: 2.04 Changed

Req: GET /bnd
Res: 2.05 Content (application/link-format)
    <coap://sensor.example.com/s/light>;
        rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"

Req: DELETE /bnd
Res: 2.04 Changed

5.9. Resource Observation Attributes

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [I-D.ietf-core-observe] MAY be used to observe any changes in a resource, and receive asynchronous notifications as a result. In addition, a set of query string parameters are defined here to allow a client to control how often a client is interested in receiving notifications and how much a resource value should change for the new representation to be interesting. These query parameters are described in the following table. A resource using an interface description defined in this specification and marked as Observable in its link description SHOULD support these observation parameters. The Change Step parameter can only be supported on resources with an atomic numeric value.

These query parameters MUST be treated as resources that are read using GET and set using PUT, and MUST NOT be included in the Observe request. Multiple parameters MAY be set at the same time by including the values in the query string of a PUT. Before being set, these parameters have no default value.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Parameter</th>
<th>Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Period</td>
<td>/{resource}?pmin</td>
<td>xsd:integer (&gt;0)</td>
</tr>
<tr>
<td>Maximum Period</td>
<td>/{resource}?pmax</td>
<td>xsd:integer (&gt;0)</td>
</tr>
<tr>
<td>Change Step</td>
<td>/{resource}?st</td>
<td>xsd:decimal (&gt;0)</td>
</tr>
<tr>
<td>Less Than</td>
<td>/{resource}?lt</td>
<td>xsd:decimal</td>
</tr>
<tr>
<td>Greater Than</td>
<td>/{resource}?gt</td>
<td>xsd:decimal</td>
</tr>
</tbody>
</table>
Minimum Period: When present, the minimum period indicates the minimum time to wait (in seconds) before sending a new synchronization message (even if it has changed). In the absence of this parameter, the minimum period is up to the notifier.

Maximum Period: When present, the maximum period indicates the maximum time in seconds between two consecutive synchronisation messages (regardless if it has changed). In the absence of this parameter, the maximum period is up to the notifier. The maximum period MUST be greater than the minimum period parameter (if present).

Change Step: When present, the change step indicates how much the value of a resource SHOULD change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification SHOULD be between pmin and pmax.

Greater Than: When present, Greater Than indicates the upper limit value the resource value SHOULD cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Greater Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

Less Than: When present, Less Than indicates the lower limit value the resource value SHOULD cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Less Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

5.10. Future Interfaces

It is expected that further interface descriptions will be defined in this and other specifications. Potential interfaces to be considered for this specifications include:

Collection: This resource would be a container that allows sub-resources to be added or removed.

5.11. WADL Description

This section defines the formal Web Application Description Language (WADL) definition of these CoRE interface descriptions.

<?xml version="1.0" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:senml="urn:ietf:params:xml:ns:senml">
<grammars>
  <include href="http://tools.ietf.org/html/draft-jennings-senml"/>
</grammars>
<doc title="CoRE Interfaces"/>

<resource_type id="s">
  <doc title="Sensor resource type"/>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#observe-cancel"/>
  <method href="#getattr"/>
  <method href="#setattr"/>
</resource_type>

<resource_type id="p">
  <doc title="Parameter resource type"/>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#observe-cancel"/>
  <method href="#getattr"/>
  <method href="#setattr"/>
  <method href="#update"/>
</resource_type>

<resource_type id="rp">
  <doc title="Read-only Parameter resource type"/>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#observe-cancel"/>
  <method href="#getattr"/>
  <method href="#setattr"/>
</resource_type>

<resource_type id="a">
  <doc title="Actuator resource type"/>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#observe-cancel"/>
  <method href="#getattr"/>
  <method href="#setattr"/>
  <method href="#update"/>
  <method href="#toggle"/>
</resource_type>

<resource_type id="ll">
  <doc title="Link List type"/>
  <method href="#listLinks"/>
</resource_type>

<resource_type id="b">
  <doc title="Batch of sub-resources type">
The methods read, observe, update and toggle are applied to each sub-
resource of the requested resource that supports it. Mixed sub-resource types can be supported.</doc>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#observe-cancel"/>
  <method href="#getattr"/>
  <method href="#setattr"/>
  <method href="#update"/>
  <method href="#toggle"/>
  <method href="#listLinks"/>
</resource_type>

<resource_type id="lb">
  <doc title="Linked Batch resource type">. The methods read, obervableRead, update and toggle are applied to each linked resource of the requested resource that supports it. Mixed linked resource types can be supported.</doc>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#observe-cancel"/>
  <method href="#getattr"/>
  <method href="#setattr"/>
  <method href="#update"/>
  <method href="#listLinks"/>
  <method href="#appendLinks"/>
  <method href="#clearLinks"/>
</resource_type>

<resource_type id="bnd">
  <doc title="Binding table resource type">A modifiable list of links. Each link MUST have the relation type "boundTo".</doc>
  <method href="#listLinks"/>
  <method href="#appendLinks"/>
  <method href="#clearLinks"/>
</resource_type>

<method id="read" name="GET">
  <doc>Retrieve the value of a sensor, an actuator or a parameter. Both HTTP and CoAP support this method.</doc>
  <request>
<request>
  <response status="200">
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </response>
  <response status="2.05">
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </response>
</method>

<method id="observe" name="GET">
  <doc>Observe the value of a sensor, an actuator or a parameter. Only CoAP supports this method since it requires the CoRE Observe mechanism.</doc>
  <request>
    <param name="observe" style="header" type="xsd:integer">
      <option value = 0/>
    </param>
  </request>
  <response status="2.05">
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </response>
</method>

<method id="observe-cancel" name="GET">
  <doc>Cancel observation in progress. Only CoAP supports this method since it requires the CoRE Observe mechanism.</doc>
  <request>
    <param name="observe" style="header" type="xsd:integer">
      <option value = 1/>
    </param>
  </request>
  <response status="2.05">
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </response>
</method>

<method id="update" name="PUT">
  <doc>Control the actuator or update a parameter with a new value or command. Both HTTP and CoAP support this method.</doc>
  <request>
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="getattr" name="GET">
  <doc>Retrieve the observe attributes associated with a resource. Both HTTP and CoAP support this method.</doc>
  <request>
    <doc>This request MUST contain an Accept option with application/link-format when the resource supports other GET methods.</doc>
    <representation mediaType="application/link-format"/>
  </request>
  <response status="200"/>
  <response status="2.05"/>
</method>

<method id="setattr" name="PUT">
  <doc>Set the values of some or all of the observe attributes associated with a resource. Both HTTP and CoAP support this method.</doc>
  <request>
    <param name="pmin" style="query" type="xsd:integer"/>
    <param name="pmax" style="query" type="xsd:integer"/>
    <param name="lt" style="query" type="xsd:decimal"/>
    <param name="gt" style="query" type="xsd:decimal"/>
    <param name="st" style="query" type="xsd:decimal"/>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>
<method id="toggle" name="POST">
  <doc>Toggle the values of actuator resources. Both HTTP and CoAP support this method.</doc>
  <request>
    <doc>The toggle function is only applicable if the request is empty.</doc>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="listLinks" name="GET">
  <doc>Retrieve the list of Web links associated to a resource. Both HTTP and CoAP support this method.</doc>
  <request>
    <doc>This request MUST contain an Accept option with application/link-format when the resource supports other GET methods.</doc>
  </request>
  <response status="200"/>
    <representation mediaType="application/link-format"/>
  </response>
  <response status="2.05"/>
    <representation mediaType="application/link-format"/>
  </response>
</method>

<method id="appendLinks" name="POST">
  <doc>Append new Web links to a resource which is a collection of links. Both HTTP and CoAP support this method.</doc>
  <request>
    <representation mediaType="application/link-format"/>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="clearLinks" name="DELETE">
  <doc>Clear all Web Links in a resource which is a collection of links. Both HTTP and CoAP support this method.</doc>
  <request>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

</application>
6. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this document. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service.

7. IANA Considerations

The interface description types defined require registration.

The new link relation type "boundto" requires registration.

8. Acknowledgments

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this document.

9. Changelog

Changes from -02 to -03

- Added lt and gt to binding format section.
- Added pmin and pmax observe parameters to Observation Attributes.
- Changed the definition of lt and gt to limit crossing.
- Added definitions for getattr and setattr to WADL.
- Added getattr and setattr to observable interfaces.
- Removed query parameters from Observe definition.
- Added observe-cancel definition to WADL and to observable interfaces.
Changes from -01 to -02
  o Updated the date and version, fixed references.
  o Removed pmin and pmax observe parameters [Ticket #336]
Changes from -00 to WG Document -01
  o Improvements to the Function Set section.
Changes from -05 to WG Document -00
  o Updated the date and version.
Changes from -04 to -05
  o Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.
Changes from -03 to -04
  o Draft refresh
Changes from -02 to -03
  o Added Bindings
  o Updated all rt= and if= for the new Link Format IANA rules
Changes from -01 to -02
  o Defined a Function Set and its guidelines.
  o Added the Link List interface.
  o Added the Linked Batch interface.
  o Improved the WADL interface definition.
  o Added a simple profile example.

10. References
10.1. Normative References


10.2. Informative References


Appendix A. Profile example

The following is a short definition of simple profile. This simplistic profile is for use in the examples of this document.

<table>
<thead>
<tr>
<th>Function Set</th>
<th>Root Path</th>
<th>RT</th>
<th>IF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Description</td>
<td>/d</td>
<td>simple.dev</td>
<td>core.ll</td>
</tr>
<tr>
<td>Sensors</td>
<td>/s</td>
<td>simple.sen</td>
<td>core.b</td>
</tr>
<tr>
<td>Actuators</td>
<td>/a</td>
<td>simple.act</td>
<td>core.b</td>
</tr>
</tbody>
</table>

List of Function Sets
### Device Description Function Set

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
<th>RT</th>
<th>IF</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>/d/name</td>
<td>simple.dev.n</td>
<td>core.p</td>
<td>xsd:string</td>
</tr>
<tr>
<td>Model</td>
<td>/d/model</td>
<td>simple.dev.mdl</td>
<td>core.rp</td>
<td>xsd:string</td>
</tr>
</tbody>
</table>

### Sensors Function Set

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
<th>RT</th>
<th>IF</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light</td>
<td>/s/light</td>
<td>simple.sen.lt</td>
<td>core.s</td>
<td>xsd:decimal (lux)</td>
</tr>
<tr>
<td>Humidity</td>
<td>/s/humidity</td>
<td>simple.sen.hum</td>
<td>core.s</td>
<td>xsd:decimal (%RH)</td>
</tr>
<tr>
<td>Temperature</td>
<td>/s/temp</td>
<td>simple.sen.tmp</td>
<td>core.s</td>
<td>xsd:decimal (degC)</td>
</tr>
</tbody>
</table>

### Actuators Function Set

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
<th>RT</th>
<th>IF</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED</td>
<td>/a/({#})/led</td>
<td>simple.act.led</td>
<td>core.a</td>
<td>xsd:boolean</td>
</tr>
</tbody>
</table>

Authors’ Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
FINLAND

Phone: +1-408-203-9434
Email: zach.shelby@arm.com
Matthieu Vial
Schneider-Electric
Grenoble,
FRANCE

Phone: +33 (0) 47657 6522
Email: matthieu.vial@schneider-electric.com

Michael Koster
ARM
150 Rose Orchard
San Jose 95134
FINLAND

Phone: 
Email: michael.koster@arm.com
Representing CoRE Formats in JSON and CBOR

draft-ietf-core-links-json-03

Abstract

JavaScript Object Notation, JSON (RFC7159) is a text-based data format which is popular for Web based data exchange. Concise Binary Object Representation, CBOR (RFC7049) is a binary data format which has been optimized for data exchange for the Internet of Things (IoT). For many IoT scenarios, CBOR formats will be preferred since it can help decrease transmission payload sizes as well as implementation code sizes compared to other data formats.

Web Linking (RFC5988) provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format (RFC6690). Outside of constrained environments, it may be useful to represent these collections of Web links in JSON, and similarly, inside constrained environments, in CBOR. This specification defines a common format for this.

Group Communication for the Constrained Application Protocol (RFC7390) defines a number of JSON formats for controlling communication between groups of nodes employing the Constrained Application Protocol (CoAP). In a similar vein, this specification defines CBOR variants of these formats.

Status of This Memo

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

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

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

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

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

Table of Contents

1. Introduction .............................................. 3
   1.1. Objectives ........................................ 3
   1.2. Terminology ....................................... 4
2. Web Links in JSON and CBOR ............................... 4
   2.1. Background ....................................... 4
   2.2. Information Model ................................. 5
   2.3. Additional Encoding Step for CBOR ................. 6
   2.4. Examples .......................................... 7
       2.4.1. Link Format to CBOR Example ............... 8
       2.4.2. Link Format in JSON to CBOR Example ...... 9
3. Group Communication Management Objects in CBOR ........ 11
   3.1. Background ....................................... 11
   3.2. Information Model ................................. 11
   3.3. Mapping .......................................... 11
   3.4. Group Communication Example ..................... 12
4. IANA Considerations ...................................... 14
5. Security Considerations .................................. 15
6. Acknowledgements ........................................ 15
7. References .............................................. 16
   7.1. Normative References ............................. 16
   7.2. Informative References ........................... 16
Appendix A. Implementation .................................. 17
Authors’ Addresses ......................................... 17
1. Introduction

Web Linking [RFC5988] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252].

The JavaScript Object Notation (JSON) [RFC7159] is a lightweight, text-based, language-independent data interchange format. JSON is popular in the Web development environment as it is easy for humans to read and write.

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

When converting between a bespoke syntax such as that defined by [RFC6690] and JSON or CBOR, many small decisions have to be made. If left without guidance, it is likely that a number of slightly incompatible dialects will emerge. This specification defines a common approach for translating between the CoRE-specific bespoke formats, JSON and CBOR formats. Where applicable, mapping from other formats (e.g. CoRE Link Format) into JSON or CBOR is also described.

This specification defines a common format for representing CoRE Web Linking in JSON and CBOR, as well as the various JSON formats for controlling CoRE group communication [RFC7390], in CBOR.

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

1.1. Objectives

This specification has been designed based on the following objectives:

- Canonical mapping
  - lossless round-tripping with [RFC6690] and between JSON and CBOR
  - but not trying for bit-preserving (DER-style) round-tripping
The simplest thing that could possibly work

* Do not cater for RFC 5988 complications caused by HTTP header character set issues [RFC2047]

Consider other work that has links in JSON, e.g.: JSON-LD, JSON-Reference [I-D.pbryan-zyp-json-ref]

* Do not introduce unmotivated differences

1.2. Terminology

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

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

CoAP: Constrained Application Protocol [RFC7252]

CBOR: Concise Binary Object Representation [RFC7049]

CoRE: Constrained RESTful Environments, the field of work underlying [RFC6690], [RFC7049], [RFC7252], and [RFC7390]

IoT: Internet of Things

JSON: JavaScript Object Notation [RFC7159]

The objective of the JSON and CBOR mappings defined in this document is to contain information of the formats specified in [RFC5988] and [RFC6690]. This specification therefore uses the names of the ABNF productions used in those documents.

2. Web Links in JSON and CBOR

2.1. Background

Web Linking [RFC5988] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252] and in
2.2. Information Model

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

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

We straightforwardly map:

- the outer collection to an array of links;
- each link to a JSON object or CBOR map, mapping attribute names to attribute values.

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a JSON name/value pair (member). The name is a string representation of the parameter or attribute name (as in "parmname"), the value is a string representation of the parameter or attribute value ("ptoken" or "quoted-string"). "quoted-string" productions are parsed (i.e., the outer quotes removed and the backslash constructions evaluated) as defined in [RFC6690] and its referenced documents, before placing them in JSON strings (where they may gain back additional decorations such as backslashes as defined in [RFC7159]).

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

If a Link attribute ("parmname") is present more than once in a "link-value", its values are then represented as a JSON array of JSON string values; this array becomes the value of the JSON name/value pair where the attribute name is the JSON name. Attributes occurring just once MUST NOT be represented as JSON arrays but MUST be directly represented as JSON strings. (Note that [RFC6690] has cut down on the use of repeated parameter names; they are still allowed by [RFC5988] though. No attempt has been made to decode the possibly space-separated values for rt=, if=, and rel= into JSON arrays.)

The URI-Reference is represented as a name/value pair with the name "href" and the URI-Reference as the value. (Rationale: This usage is consistent with the use of "href" as a query parameter for link-
format query filtering and with link-format reserving the link parameter "href" specifically for this use [RFC6690]).

The resulting structure can be represented in CDDL [I-D.greevenbosch-appsawg-cbor-cddl] as:

```plaintext
links = [ * link
  link = {
    href: tstr    ; resource URI
    * tstr => tstr / true
  }
```

Figure 1: CoRE Link Format Data Model

2.3. Additional Encoding Step for CBOR

The above specification for JSON could be used as is for the CBOR encoding as well. However, to further reduce message sizes, it is beneficial to perform an extra encoding step, and encode "href" and some commonly occurring attribute names as small integers.

The substitution is summarized below:

```
+----------+---------------+
| name     | encoded value |
+----------+---------------+
| href     | 1             |
| rel      | 2             |
| anchor   | 3             |
| rev      | 4             |
| hreflang | 5             |
| media    | 6             |
| title    | 7             |
| type     | 8             |
| rt       | 9             |
| if       | 10            |
| sz       | 11            |
| ct       | 12            |
| obs      | 13            |
```

Table 1: Integer Encoding of common attribute names

** TO DO: Is this the right list of attribute names? **

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

This leads to the following CDDL representation for the CBOR encoding:

```cddl
links = [* link]
link = {
    href: tstr    ; resource URI
    * label => tstr / true
}
label = tstr / &{
    href: 1,    rel: 2,        anchor: 3,
    rev: 4,     hreflang: 5,   media: 6,
    title: 7,   type: 8,       rt: 9,
    if: 10,     sz: 11,        ct: 12,
    obs: 13,    
}
```

Figure 2: CoRE Link Format Data Model (CBOR)

2.4. Examples

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

Figure 3: Example from page 15 of [RFC6690]

The link-format document in Figure 3 becomes (321 bytes):

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

(More examples to be added.)
2.4.1. Link Format to CBOR Example

This example shows conversion from link format to CBOR format.

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

```json
[1: "/sensors", 12: "40", 7: "Sensor Index"],
```

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"
 31                          # 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"
 31                          # map(# data item pairs:3)
    01                          # unsigned integer(value:1,"href")
    6e                          # text string(14 bytes)
      2f73656e736f72732f6c 69766974        # "/sensors/light"
 0a                          # unsigned integer(value:9,"rt")
 69                          # text string(9 bytes)
    6c696768742d6c7578         # "light-lux"
 0a                          # unsigned integer(value:10,"if")
```

[Page 8]
2.4.2. Link Format in JSON to CBOR Example

The JSON example from Section 2.4 becomes:

```
85                                # array(number of data items:5)
  a3                             # map(# data item pairs:3)
    01                          # unsigned integer(value:1, "href")
    68                          # text string(8 bytes)
      2f73656e736f72732f74     # "/sensors"
    0c                          # unsigned integer(value:12, "ct")
    18                          # unsigned integer(value:40)
    07                          # unsigned integer(value:7, "title")
    6c                          # text string(12 bytes)
      53656e736f7220496e617465 # "Sensor Index"
  a3                             # map(# data item pairs:3)
    01                          # unsigned integer(value:1, "href")
```

Figure 4: Web Links Encoded in CBOR
6d                          # text string(13 bytes)
 2f73656e736f72732f74
656d70                      # "/sensors/temp"
09                          # unsigned integer(value:9,"rt")
6d                          # text string(13 bytes)
74656d70657261747572
656d63
0a                          # "temperature-c"
66                          # unsigned integer(value:10,"if")
73656e736f72
09                          # text string(14 bytes)
2f73656e736f72732f6c
69676874
09                          # unsigned integer(value:9,"rt")
6c696768742d6c7578
0a                          # "light-lux"
66                          # unsigned integer(value:10,"if")
73656e736f72
03                          # text string(35 bytes)
2f73656e736f72732f74
656d70
02                          # http://www.example.com/sensors/t123"
6b                          # unsigned integer(value:3,"anchor")
646573637269626564273
2f74313233
03                          # text string(13 bytes)
2f73656e736f72732f74
656d70
02                          # "/sensors/temp"
6b                          # unsigned integer(value:2,"rel")
646573637269626564273
2f74313233
03                          # text string(11 bytes)
2f73656e736f72732f74
656d70
02                          # "describedby"
6b                          # map(# data item pairs:3)
646573637269626564273
2f74313233
03                          # map(# data item pairs:3)
2f73656e736f72732f74
656d70
02                          # "/t"
6b                          # "/sensors/temp"
646573637269626564273
2f74313233
03                          # text string(35 bytes)
2f73656e736f72732f74
656d70
02                          # "alternate"
3. Group Communication Management Objects in CBOR

3.1. Background

The CoAP Group Communications specification [RFC7390] defines group management objects in JSON format. These objects are used to represent IP multicast group information for CoAP endpoints. See [I-D.ietf-core-resource-directory] for more examples of using these objects.

3.2. Information Model

This section discusses the information model underlying the CoAP Group Communication management object payload.

A group membership JSON object contains one or more key/value pairs, and represents a single IP multicast group membership for the CoAP endpoint. Each key/value pair is encoded as a member of the JSON object, where the key is the member name and the value is the member’s value.

The information model of the CoAP Group Communication management object can be summarized below:

```plaintext
collection = { * index => membership }
index = tstr .regexp "[A-Za-z0-9]{1,2}"
membership = {
  ? n: groupname,
  ? a: groupaddress,
}
groupname = tstr ; host [:" port]
groupaddress = tstr ; IPv4address [:" port ]
                         ; / "[IPv6address "]
                         [":" port ]
```

3.3. Mapping

The objective of the mapping defined in this section is to map information from the JSON formats specified in [RFC7390] into CBOR format, using the rules of Section 4.2 of [RFC7049].
3.4. Group Communication Example

```json
{
    "8": { "a": "[ff15::4200:f7fe:ed37:14ca]" },
    "11": { "n": "sensors.floor1.west.bldg6.example.com",
             "a": "[ff15::4200:f7fe:ed37:25cb]" },
    "12": { "n": "All-Devices.floor1.west.bldg6.example.com",
             "a": "[ff15::4200:f7fe:ed37:abcd]:4567" }
}
```

Figure 7: Example from section 2.6.2.4 of [RFC7390] becomes:
Figure 8: Group Communication Management Object Encoded in CBOR
TO DO: Should the IP address/port number information be represented in a more compact way?

4. IANA Considerations

This specification registers the following additional Internet Media Types:

Type name: application

Subtype name: link-format+json

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+json" media type are required to conform to the "application/json" Media Type and are therefore subject to the same encoding considerations specified in [RFC7159], Section 11.

Security considerations: As defined in this specification

Published specification: This specification.

Applications that use this media type: None currently known.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): TEXT

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

Intended usage: COMMON

Change controller: IESG

and
Type name: application
Subtype name: link-format+cbor
Required parameters: None
Optional parameters: None
Encoding considerations: Resources that use the "application/link-format+cbor" media type are required to conform to the "application/cbor" Media Type and are therefore subject to the same encoding considerations specified in [RFC7049], Section 7.
Security considerations: As defined in this specification
Published specification: This specification.
Applications that use this media type: None currently known.
Additional information:
  Magic number(s): N/A
  File extension(s): N/A
  Macintosh file type code(s): CBOR
Person & email address to contact for further information:
Kepeng Li &lt;kepeng.lkp@alibaba-inc.com&gt;
Intended usage: COMMON
Change controller: IESG
5. Security Considerations
The security considerations of [RFC6690], [RFC7049] and [RFC7159] apply.
(TBD.)
6. Acknowledgements
(TBD.)
Special thanks to Bert Greevenbosch who was an author on the initial version of a contributing document, as well as the original author on the CDDL notation.
7. References

7.1. Normative References


7.2. Informative References


Appendix A. Implementation

This appendix provides a simple reference implementation of the mapping between CoRE link format and Links-in-JSON.

(TBD - the reference implementation was used to create the above examples, but I still have to clean it up for readability and paste it in at 69 columns max.)

Authors’ Addresses

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

Email: kepeng.lkp@alibaba-inc.com

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

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

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

Phone: +49-421-218-63921
Email: cabo@tzi.org
Observing Resources in CoAP
draft-ietf-core-observe-16

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

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

1.1. Background

The Constrained Application Protocol (CoAP) [RFC7252] is intended to provide RESTful services [REST] not unlike HTTP [RFC7230] while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes [RFC7228].

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

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

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

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

1.2. Protocol Overview

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

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

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

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

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

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

1.3. Consistency Model

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

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

The protocol addresses this issue as follows:

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

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

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

1.4. Observable Resources

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

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

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

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

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

- \(<\text{coap://server/?query=select+avg(temperature)+from+Sensor.window:}
  \text{time(30sec)}>\), 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.

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

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

Table 1: The Observe Option

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

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

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

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

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

3. Client-side Requirements

3.1. Request

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

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

3.2. Notifications

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

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

3.3. Caching

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

3.3.1. Freshness

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

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

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

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

3.3.2. Validation

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

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

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

3.4. Reordering

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

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

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

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

3.5. Transmission

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

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

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

3.6. Cancellation

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

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

4. Server-side Requirements

4.1. Request

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

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

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

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

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

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

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

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

4.3. Caching

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

4.3.1. Freshness

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

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

4.3.2. Validation

A client can include a set of entity-tags in its request using the ETag Option. When a 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:

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

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

9. Acknowledgements

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

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

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

10. References
10.1. Normative References


10.2. Informative References


### Appendix A. Examples

#### A.1. Client/Server Examples

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

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

Figure 4: The client re-registers after Max-Age ends
Figure 5: The client re-registers and gives the server the opportunity to select a stored response
<table>
<thead>
<tr>
<th>t</th>
<th>Observed State</th>
<th>CLIENT State</th>
<th>SERVER State</th>
<th>Actual State</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>19.7 Cel</td>
<td></td>
<td></td>
<td>19.7 Cel</td>
</tr>
<tr>
<td>89</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td></td>
<td></td>
<td>&lt;-----+</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td></td>
<td></td>
<td>2.05</td>
<td>19.3 Cel</td>
</tr>
<tr>
<td>93</td>
<td>19.3 Cel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td></td>
<td></td>
<td>Observe: 91</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td></td>
<td></td>
<td>Max-Age: 15</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td>Payload: &quot;19.3 Cel&quot;</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td></td>
<td></td>
<td>-- - -&gt;</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
<td></td>
<td></td>
<td>19.0 Cel</td>
</tr>
<tr>
<td>105</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>19.3 Cel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>(stale)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>CLIENT</th>
<th>PROXY</th>
<th>SERVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⮕</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⮕ GET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⮕ ⮕</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⮕ ⮕ ⮕ GET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⮕ ⮕ ⮕ ⮕ ⮕ GET</td>
<td>0x41015fb8</td>
<td></td>
</tr>
<tr>
<td>⮕ ⮕ ⮕ ⮕ ⮕ ⮕</td>
<td>0x1a</td>
<td></td>
</tr>
<tr>
<td>⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ Header: GET</td>
<td>0x41015fb8</td>
<td></td>
</tr>
<tr>
<td>⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ Token: 0x1a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ Uri-Host: sensor.example</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ Uri-Path: status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ Observe: 0 (register)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ ⮕ }
Figure 8: A client observes a resource through a proxy
Appendix B. Changelog

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

Changes from ietf-15 to ietf-16:

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

Changes from ietf-14 to ietf-15:

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

Changes from ietf-13 to ietf-14:

- Updated references.

Changes from ietf-12 to ietf-13:

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

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

- Removed the 7.31 Code for cancellation.

Changes from ietf-11 to ietf-12:

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

- Made the algorithm for superseding an outstanding transmission OPTIONAL.

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

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

Changes from ietf-10 to ietf-11:
Internet-Draft         Observing Resources in CoAP         December 2014

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

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

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

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

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

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

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

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

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

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

Changes from ietf-06 to ietf-07:

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

Changes from ietf-05 to ietf-06:

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

Changes from ietf-04 to ietf-05:

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

- Relaxed requirements when sending a Reset message in reply to non-confirmable notifications.

- Added an implementation note about careless GET requests (#184).

- Updated examples.

Changes from ietf-03 to ietf-04:

- Removed the "Max-OFE" Option.

- Allowed a Reset message in reply to non-confirmable notifications.

- Added a section on cancellation.

- Updated examples.

Changes from ietf-02 to ietf-03:

- Separated client-side and server-side requirements.

- Fixed uncertainty if client is still on the list of observers by introducing a liveliness model based on Max-Age and a new option called "Max-OFE" (#174).

- Simplified the text on message reordering (#129).

- Clarified requirements for intermediaries.

- Clarified the combination of blockwise transfers with notifications (#172).

- Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.

- Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:

- Removed the requirement of periodic refreshing (#126).

- The new "Observe" Option replaces the "Lifetime" Option.

- Introduced a new mechanism to detect message reordering.
- 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 Resource Directory
draft-ietf-core-resource-directory-04

Abstract

In many M2M applications, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources. This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resources descriptions. Furthermore, new link attributes useful in conjunction with an RD are defined.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

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

1. Introduction ........................................... 3
2. Terminology ............................................. 4
3. Architecture and Use Cases ............................ 5
   3.1. Use Case: Cellular M2M ............................ 6
   3.2. Use Case: Home and Building Automation .......... 7
   3.3. Use Case: Link Catalogues ........................ 7
4. Simple Directory Discovery ............................. 8
   4.1. Finding a Directory Server ....................... 9
   4.2. Third-party registration .......................... 10
5. Resource Directory Function Set ...................... 10
   5.1. Discovery ........................................... 10
   5.2. Registration ........................................ 12
   5.3. Update .............................................. 14
   5.4. Removal ............................................. 16
   5.5. Read Endpoint Links ................................ 17
6. Group Function Set ..................................... 18
   6.1. Register a Group .................................. 18
   6.2. Group Removal .................................... 20
7. RD Lookup Function Set ................................ 21
8. New Link-Format Attributes ........................... 25
   8.1. Resource Instance attribute ‘ins’ ............... 25
   8.2. Export attribute ‘exp’ ............................. 25
9. DNS-SD Mapping ........................................ 26
   9.1. DNS-based Service discovery ...................... 26
   9.2. mapping ins to <Instance> ........................ 27
   9.3. Mapping rt to <ServiceType> .................... 28
   9.4. Domain mapping .................................... 28
   9.5. TXT Record key=value strings ................... 28
   9.6. Importing resource links into DNS-SD .......... 29
10. Security Considerations .............................. 30
    10.1. Endpoint Identification and Authentication ... 30
    10.2. Access Control .................................... 30
    10.3. Denial of Service Attacks ...................... 30
11. IANA Considerations .................................. 31
    11.1. Resource Types .................................... 31
    11.2. Link Extension ................................... 31
The work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g., 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. This specification however only describes how to discover resources from the web server that hosts them by requesting "/.well-known/core". In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources.

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

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The term "byte" is used in its now customary sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

- **Resource Directory**: A web entity that stores information about web resources and implements the REST interfaces defined in this specification for registration and lookup of those resources.

- **Domain**: In the context of a Resource Directory, a domain is a logical grouping of endpoints. This specification assumes that the list of Domains supported by an RD is pre-configured by that RD. When a domain is exported to DNS, the domain value equates to the DNS domain name.

- **Group**: In the context of a Resource Directory, a group is a logical grouping of endpoints for the purpose of group communications. All groups within a domain are unique.

- **Endpoint**: Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the Resource Directory. An endpoint is identified by its endpoint name, which is included during registration, and is unique within the associated domain of the registration.
3. Architecture and Use Cases

The resource directory architecture is illustrated in Figure 1. A Resource Directory (RD) is used as a repository for Web Links [RFC5988] about resources hosted on other web servers, which are called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port (called Context), thus a physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain sets of Web Links (called resource directory entries), and for clients to lookup resources from the RD or maintain groups. Endpoints themselves can also act as clients. An RD can be logically segmented by the use of Domains. The domain an endpoint is associated with can be defined by the RD or configured by an outside entity. This information hierarchy is shown in Figure 2.

Endpoints are assumed to proactively register and maintain resource directory entries on the RD, which are soft state and need to be periodically refreshed. An endpoint is provided with interfaces to register, update and remove a resource directory entry. Furthermore, a mechanism to discover an RD using the CoRE Link Format is defined. It is also possible for an RD to proactively discover Web Links from endpoints and add them as resource directory entries. A lookup interface for discovering any of the Web Links held in the RD is provided using the CoRE Link Format.

```
+-----+  +-----+  +-----+  +-----+  +-----+  +-----+
| EP  |----| EP  |----| RD  |----| RD  |----| Client |
|-----|    |-----|    |-----|    |-----|    |
+-----+  +-----+  +-----+  +-----+  +-----+
```

Figure 1: The resource directory architecture.
3.1. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded air interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of view, the ambition is to design horizontal solutions that can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo-like solutions. One of the crucial enablers of such design is the ability to discover resources (machines -- endpoints) capable of providing required information at a given time or acting on instructions from the end users.

In a typical scenario, during a boot-up procedure (and periodically afterwards), the machines (endpoints) register with a Resource Directory (for example EPs installed on vehicles enabling tracking of their position for fleet management purposes and monitoring environment parameters) hosted by the mobile operator or somewhere else in the network, periodically describing its own capabilities. Due to the usual network configuration of mobile networks, the EPs attached to the mobile network may not always be efficiently reachable. Therefore, a remote server is usually used to provide proxy access to the EPs. The address of each (proxy) endpoint on this server is included in the resource description stored in the RD. The users, for example mobile applications for

Figure 2: The resource directory information hierarchy.
environment monitoring, contact the RD, look-up the endpoints capable
of providing information about the environment using appropriate set
of link parameters, obtain information on how to contact them (URLs
of the proxy server) and then initiate interaction to obtain
information that is finally processed, displayed on the screen and
usually stored in a database. Similarly, fleet management systems
provide the appropriate link parameters to the RD to look-up for EPs
deployed on the vehicles the application is responsible for.

3.2. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the
use of M2M web services. The discovery requirements of these
applications are demanding. Home automation usually relies on run-
time discovery to commission the system, whereas in building
automation a combination of professional commissioning and run-time
discovery is used. Both home and building automation involve peer-
to-peer interactions between endpoints, and involve battery-powered
sleeping devices.

The exporting of resource information to other discovery systems is
also important in these automation applications. In home automation
there is a need to interact with other consumer electronics, which
may already support DNS-SD, and in building automation larger
resource directories or DNS-SD covering multiple buildings.

3.3. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge
beforehand of who is going to consume the data. Resource Directory
can be used to hold links about resources and services hosted
anywhere to make them discoverable by a general class of
applications.

For example, environmental and weather sensors that generate data for
public consumption may provide the data to an intermediary server, or
broker. Sensor data are published to the intermediary upon changes
or at regular intervals. Descriptions of the sensors that resolve to
links to sensor data may be published to a Resource Directory.
Applications wishing to consume the data can use the Resource
Directory lookup function set to discover and resolve links to the
desired resources and endpoints. The Resource Directory service need
not be coupled with the data intermediary service. Mapping of
Resource Directories to data intermediaries may be many-to-many.

Metadata in link-format, link-format+cbor, or link-format+json
representations are supplied by Resource Directories, which may be
internally stored as triples, or relation/attribute pairs providing
metadata about resource links. External catalogs that are represented in other formats may be converted to link-format, link-format+json, or link-format+cbor for storage and access by Resource Directories. Since it is common practice for these to be URN encoded, simple and lossless structural transforms will generally be sufficient to store external metadata in Resource Directories.

The additional features of Resource Directory allow domains to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Resource groups may defined to allow batched reads from multiple resources.

4. Simple Directory Discovery

Not all endpoints hosting resources are expected to know how to implement the Resource Directory Function Set (see Section 5) and thus explicitly register with a Resource Directory (or other such directory server). Instead, simple endpoints can implement the generic Simple Directory Discovery approach described in this section. An RD implementing this specification MUST implement Simple Directory Discovery. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the endpoint makes available the hosted resources that it wants to be discovered, as links on its "/.well-known/core" interface as specified in [RFC6690].

The endpoint then finds one or more IP addresses of the directory server it wants to know about its resources as described in Section 4.1.

An endpoint that wants to make itself discoverable occasionally sends a POST request to the "/.well-known/core" URI of any candidate directory server that it finds. The body of the POST request is either

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

or

- a non-empty link-format document, which indicates the specific services that the requesting server wants to make known to the directory server.
The directory server integrates the information it received this way into its resource directory. It MAY make the information available to further directories, if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

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

<table>
<thead>
<tr>
<th>EP</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- POST /.well-known/core &quot;/sen/temp&quot;... --&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;---- 2.01 Created -------------------------</td>
<td></td>
</tr>
</tbody>
</table>

4.1. Finding a Directory Server

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

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

- specific static configuration (e.g., anycast addresses), if any,
- the ABRO option of 6LoWPAN-ND [RFC6775],
- other ND options that happen to point to servers (such as RDNSS),
- DHCPv6 options that might be defined later.

In networks with more inexpensive use of multicast, the candidate IP address may be a well-known multicast address, i.e. directory servers are found by simply sending GET requests to that well-known multicast address (see Section 5.1).

As some of these sources are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don’t work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.
4.2. Third-party registration

For some applications, even Simple Directory Discovery may be too taxing for certain very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the Resource Directory can be filled by a third device, called an installation tool. The installation tool can fill the Resource Directory from a database or other means. For that purpose the scheme, IP address and port of the registered device is indicated in the Context parameter of the registration as well.

5. Resource Directory Function Set

This section defines the REST interfaces between an RD and endpoints, which is called the Resource Directory Function Set. Although the examples throughout this section assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. In all definitions in this section, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces defined in this section.

Resource directory entries are designed to be easily exported to other discovery mechanisms such as DNS-SD. For that reason, parameters that would meaningfully be mapped to DNS SHOULD be limited to a maximum length of 63 bytes.

5.1. Discovery

Before an endpoint can make use of an RD, it must first know the RD’s IP address, port and the path of its RD Function Set. There can be several mechanisms for discovering the RD including assuming a default location (e.g. on an Edge Router in a LoWPAN), by assigning an anycast address to the RD, using DHCP, or by discovering the RD using the CoRE Link Format (see also Section 4.1). This section defines discovery of the RD using the well-known interface of the CoRE Link Format [RFC6690] as the required mechanism. It is however expected that RDs will also be discoverable via other methods depending on the deployment.

Discovery is performed by sending either a multicast or unicast GET request to "/.well-known/core" and including a Resource Type (rt) parameter [RFC6690] with the value "core.rd" in the query string. Likewise, a Resource Type parameter value of "core.rd-lookup" is used to discover the RD Lookup Function Set. Upon success, the response
will contain a payload with a link format entry for each RD discovered, with the URL indicating the root resource of the RD. When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network.

An RD implementation of this specification MUST support query filtering for the rt parameter as defined in [RFC6690].

The discovery request interface is specified as follows:

Interaction: EP -> RD

Method: GET

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

URI Template Variables:

 rt := Resource Type (optional). MAY contain the value "core.rd", "core.rd-lookup", "core.rd-group" or "core.rd*

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

The following response codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format, application/link-format+json, or application/link-format+cbor payload containing one or more matching entries for the RD resource.

Failure: 4.04 "Not Found" or 404 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: 4.00 "Bad Request" or 400 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

The following example shows an endpoint discovering an RD using this interface, thus learning that the base RD resource is, in this example, at /rd. Note that it is up to the RD to choose its base RD resource, although diagnostics and debugging is facilitated by using the base paths specified here where possible.
5.2. Registration

After discovering the location of an RD Function Set, an endpoint MAY register its resources using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690], JSON CoRE Link Format [I-D.ietf-core-links-json], or CBOR CoRE Link Format (application/link-format+cbor) along with query string parameters indicating the name of the endpoint, its domain and the lifetime of the registration. All parameters except the endpoint name are optional. It is expected that other specifications will define further parameters (see Section 11.3). The RD then creates a new resource or updates an existing resource in the RD and returns its location. An endpoint MUST use that location when refreshing registrations using this interface. Endpoint resources in the RD are kept active for the period indicated by the lifetime parameter. The endpoint is responsible for refreshing the entry within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameter does not create multiple RD entries.

The registration request interface is specified as follows:

Interaction:  EP -> RD

Method:  POST

URI Template:  /{+rd}{?ep,d,et,lt,con}

URI Template Variables:
rd := RD Function Set path (mandatory). This is the path of the RD Function Set, as obtained from discovery. An RD SHOULD use the value "rd" for this variable whenever possible.

ep := Endpoint name (mandatory). The endpoint name is an identifier that MUST be unique within a domain. The maximum length of this parameter is 63 bytes.

d := Domain (optional). The domain to which this endpoint belongs. This parameter SHOULD be less than 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain. The domain value is needed to export the endpoint to DNS-SD (see Section 9).

et := Endpoint Type (optional). The semantic type of the endpoint. This parameter SHOULD be less than 63 bytes. Optional.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

con := Context (optional). This parameter sets the scheme, address and port at which this server is available in the form scheme://host:port. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port of the register request are assumed. This parameter is mandatory when the directory is filled by a third party such as an installation tool.

Content-Type: application/link-format

Content-Type: application/link-format+json

Content-Type: application/link-format+cbor

The following response codes are defined for this interface:

Success: 2.01 "Created" or 201 "Created". The Location header MUST be included with the new resource entry for the endpoint. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration. The resource returned in the Location is only for the purpose of the Update (POST) and Removal (DELETE), and MUST NOT implement GET or PUT methods.
Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

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

| EP | --- POST /rd?ep=node1 "</sensors..." -------> |
|    |                                                 |
|    |   |                                                 |
|    |   | <-- 2.01 Created Location: /rd/4521 ----------  |
|    |   |                                                 |
|    |    |     |                                                 |
|    |    |     | Req: POST coap://rd.example.com/rd?ep=node1 |
|    |    |     | Payload: </sensors/temp>;ct=41;rt="temperature-c";if="sensor", |
|    |    |     |       </sensors/light>;ct=41;rt="light-lux";if="sensor" |
|    |    |     | Res: 2.01 Created |
|    |    |     | Location: /rd/4521 |

5.3. Update

The update interface is used by an endpoint to refresh or update its registration with an RD. To use the interface, the endpoint sends a POST request to the resource returned in the Location option in the response to the first registration. An update MAY update the lifetime or context parameters if they have changed since the last registration or update. Parameters that have not changed SHOULD NOT be included in an update. Upon receiving an update request, the RD resets the timeout for that endpoint and updates the scheme, IP address and port of the endpoint (using the source address of the update, or the context parameter if present).

An update MAY optionally add or replace links for the endpoint by including those links in the payload of the update as a CoRE Link Format document. Including links in an update message greatly increases the load on an RD and SHOULD be done infrequently. A link is replaced only if both the target URI and relation type match (see Section 10.1).

The update request interface is specified as follows:
Interaction: EP -> RD

Method: POST

URI Template: /{+location}{?lt,con}

URI Template Variables:

location := This is the Location path returned by the RD as a
result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in
seconds. Range of 60-4294967295. If no lifetime is included,
a default value of 86400 (24 hours) SHOULD be assumed.

con := Context (optional). This parameter sets the scheme,
address and port at which this server is available in the form
scheme://host:port. Optional. In the absence of this
parameter the scheme of the protocol, source IP address and
source port used to register are assumed. This parameter is
compulsory when the directory is filled by a third party such
as an installation tool.

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

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

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

The following response codes are defined for this interface:

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

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

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

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable".
Service could not perform the operation.

The following example shows an endpoint updating its registration at
an RD using this interface.
5.4. Removal

Although RD entries have soft state and will eventually timeout after their lifetime, an endpoint SHOULD explicitly remove its entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

interaction: EP -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful earlier registration.

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

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

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

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

The following examples shows successful removal of the endpoint from the RD.
5.5. Read Endpoint Links

Some endpoints may wish to manage their links as a collection, and may need to read the current set of links in order to determine link maintenance operations.

The read request interface is specified as follows:

**Interaction:** EP -> RD

**Method:** GET

**URI Template:** /{+location}{?rt,if,ct}

**URI Template Variables:**

- **location:** This is the Location path returned by the RD as a result of a successful earlier registration.

The following responses codes are defined for this interface:

**Success:** 2.05 "Content" or 200 "OK" upon success with an "application/link-format", "application/link-format+cbor", or "application/link-format+json" payload.

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

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

**Failure:** 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

The following examples show successful read of the endpoint links from the RD.
6. Group Function Set

This section defines a function set for the creation of groups of endpoints for the purpose of managing and looking up endpoints for group operations. The group function set is similar to the resource directory function set, in that a group may be created or removed. However unlike an endpoint entry, a group entry consists of a list of endpoints and does not have a lifetime associated with it. In order to make use of multicast requests with CoAP, a group MAY have a multicast address associated with it.

6.1. Register a Group

In order to create a group, a management entity used to configure groups, makes a request to the RD indicating the name of the group to create (or update), optionally the domain the group belongs to, and optionally the multicast address of the group. The registration message includes the list of endpoints that belong to that group. If an endpoint has already registered with the RD, the RD attempts to use the context of the endpoint from its RD endpoint entry. If the client registering the group knows the endpoint has already registered, then it MAY send a blank target URI for that endpoint link when registering the group. Configuration of the endpoints themselves is out of scope of this specification. Such an interface for managing the group membership of an endpoint has been defined in [RFC7390].

The registration request interface is specified as follows:

Interaction: Manager -> RD

Method: POST
URI Template: /{+rd-group}{?gp,d,con}

URI Template Variables:

rd-group := RD Group Function Set path (mandatory). This is the path of the RD Group Function Set. An RD SHOULD use the value "rd-group" for this variable whenever possible.

gp := Group Name (mandatory). The name of the group to be created or replaced, unique within that domain. The maximum length of this parameter is 63 bytes.

d := Domain (optional). The domain to which this group belongs. The maximum length of this parameter is 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain. The domain value is needed to export the endpoint to DNS-SD (see Section 9).

con := Context (optional). This parameter is used to set the IP multicast address at which this server is available in the form scheme://multicast-address:port. Optional. In the absence of this parameter no multicast address is configured. This parameter is compulsory when the directory is filled by an installation tool.

Content-Type: application/link-format

Content-Type: application/link-format+json

Content-Type: application/link-format+cbor

The following response codes are defined for this interface:

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

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

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

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

A group can be removed simply by sending a removal message to the location returned when registering the group. Removing a group MUST NOT remove the endpoints of the group from the RD.

The removal request interface is specified as follows:

Interaction: Manager -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful group registration.

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

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

Failure: 4.04 "Not Found" or 404 "Not Found". Group does not exist.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.
The following examples shows successful removal of the group from the RD.

<table>
<thead>
<tr>
<th>EP</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>--- DELETE /rd-group/412 ---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>--- 2.02 Deleted</td>
<td>--------------</td>
</tr>
</tbody>
</table>

Req: DELETE /rd-group/12
Res: 2.02 Deleted

7. RD Lookup Function Set

In order for an RD to be used for discovering resources registered with it, a lookup interface can be provided using this function set. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. Atom or HTML Link) or using more advanced interfaces (e.g. supporting context or semantic based lookup).

This function set allows lookups for domains, groups, endpoints and resources using attributes defined in the RD Function Set and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Using the Accept Option, the requester can control whether this list is returned in CoRE Link Format ("application/link-format", default) or its alternate content-formats ("application/link-format+json" or "application/link-format+cbor").

The target of these links SHOULD be the actual location of the domain, endpoint or resource, but MAY be an intermediate proxy e.g. in the case of an HTTP lookup interface for CoAP endpoints. Multiple query parameters MAY be included in a lookup, all included parameters MUST match for a resource to be returned. The character ‘*’ MAY be included at the end of a parameter value as a wildcard operator.

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: /{rd-lookup-base}/{lookup-type}?d,ep,gp,et,rt,page,count,resource-param
URI Template Variables:

rd-lookup-base := RD Lookup Function Set path (mandatory). This is the path of the RD Lookup Function Set. An RD SHOULD use the value "rd-lookup" for this variable whenever possible.

lookup-type := ("d", "ep", "res", "gp") (mandatory) This variable is used to select the kind of lookup to perform (domain, endpoint, resource, or group).

ep := Endpoint name (optional). Used for endpoint, group and resource lookups.

d := Domain (optional). Used for domain, group, endpoint and resource lookups.

page := Page (optional). Parameter can not be used without the count parameter. Results are returned from result set in pages that contains ‘count’ results starting from index (page * count).

count := Count (optional). Number of results is limited to this parameter value. If the parameter is not present, then an RD implementation specific default value SHOULD be used.

rt := Resource type (optional). Used for group, endpoint and resource lookups.

et := Endpoint type (optional). Used for group, endpoint and resource lookups.

resource-param := Link attribute parameters (optional). Any link attribute as defined in Section 4.1 of [RFC6690], used for resource lookups.

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format", "application/link-format+cbor", or "application/link-format+json" payload containing matching entries for the lookup.

Failure: 4.04 "Not Found" or 404 "Not Found" in case no matching entry is found for a unicast request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.
Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

The examples in this section assume a host with IP address FDFD:123 and a default CoAP port 61616. The following example shows a client performing a resource lookup:

Client
----- GET /rd-lookup/res?rt=temperature ----------------->

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

Req: GET /rd-lookup/res?rt=temperature
Res: 2.05 Content
<coap://[FDFD:123]:61616/temp>;rt="temperature" 

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

Client
----- GET /rd-lookup/ep?et=power-node ---------------------------->

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

Req: GET /rd-lookup/ep?et=power-node
Res: 2.05 Content
<coap://[FDFD:123]:61616>;ep="node5",
<coap://[FDFD:123]:61616>;ep="node7"

The following example shows a client performing a domain lookup:

Client
----- GET /rd-lookup/d ----------------------------------------->

<-- 2.05 Content </rd>;d=domain1</rd>;d=domain2 ----------

Req: GET /rd-lookup/d

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

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

```
Client                      RD
   ----- GET /rd-lookup/gp ---------------------------------> |
   |                                                             |
   | ----- GET /rd-lookup/gp ---------------------------------> |
   |                                                             |
   | <-- 2.05 Content </rd-group/12>;gp="lights1"; ------------- |
   |                                                             |
   |                                                             |
   | <-- 2.05 Content </rd-group/12>;gp="lights1"; ------------- |
```

Req: GET /rd-lookup/gp

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

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

```
Client                      RD
   ----- GET /rd-lookup/ep?gp=lights1-----------------------> |
   |                                                             |
   | ----- GET /rd-lookup/ep?gp=lights1-----------------------> |
   |                                                             |
   | <-- 2.05 Content <coap://[FDFD::123]:61616];ep="node1" ---- |
   |                                                             |
   |                                                             |
   | <-- 2.05 Content <coap://[FDFD::123]:61616];ep="node1" ---- |
```

Req: GET /rd-lookup/ep?gp=lights1

Res: 2.05 Content
  <coap://[FDFD::123]:61616];ep="node1",
  <coap://[FDFD::123]:61616];ep="node2",

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

----- GET /rd-lookup/gp?ep=node1 ---------------------->

< 2.05 Content <coap://[FDFD::123]:61616>;gp="lights1"; --
ep="node1" ------ |

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

Res: 2.05 Content
<coap://[FDFD::123]:61616>;gp="lights1";ep="node1",

8. New Link-Format Attributes

When using the CoRE Link Format to describe resources being discovered by or posted to a resource directory service, additional information about those resources is useful. This specification defines the following new attributes for use in the CoRE Link Format [RFC6690]:

link-extension = ( "ins" "=" quoted-string ) ; Max 63 bytes
link-extension = ( "exp" )

8.1. Resource Instance attribute 'ins'

The Resource Instance "ins" attribute is an identifier for this resource, which makes it possible to distinguish it from other similar resources. This attribute is similar in use to the <Instance> portion of a DNS-SD record (see Section 9.1, and SHOULD be unique across resources with the same Resource Type attribute in the domain it is used. A Resource Instance might be a descriptive string like "Ceiling Light, Room 3", a short ID like "AF39" or a unique UUID or iNumber. This attribute is used by a Resource Directory to distinguish between multiple instances of the same resource type within the directory.

This attribute MUST be no more than 63 bytes in length. The resource identifier attribute MUST NOT appear more than once in a link description.

8.2. Export attribute 'exp'

The Export "exp" attribute is used as a flag to indicate that a link description MAY be exported by a resource directory to external directories.
The CoRE Link Format is used for many purposes between CoAP endpoints. Some are useful mainly locally, for example checking the observability of a resource before accessing it, determining the size of a resource, or traversing dynamic resource structures. However, other links are very useful to be exported to other directories, for example the entry point resource to a functional service.

9. DNS-SD Mapping

CoRE Resource Discovery is intended to support fine-grained discovery of hosted resources, their attributes, and possibly other resource relations [RFC6690]. In contrast, service discovery generally refers to a coarse-grained resolution of an endpoint’s IP address, port number, and protocol.

Resource and service discovery are complementary in the case of large networks, where the latter can facilitate scaling. This document defines a mapping between CoRE Link Format attributes and DNS-Based Service Discovery [RFC6763] fields that permits discovery of CoAP services by either means.

9.1. DNS-based Service discovery

DNS-Based Service Discovery (DNS-SD) defines a conventional method of configuring DNS PTR, SRV, and TXT resource records to facilitate discovery of services (such as CoAP servers in a subdomain) using the existing DNS infrastructure. This section gives a brief overview of DNS-SD; see [RFC6763] for a detailed specification.

DNS-SD service names are limited to 255 octets and are of the form:

Service Name = <Instance>.<ServiceType>.<Domain>.

The service name is the label of SRV/TXT resource records. The SRV RR specifies the host and the port of the endpoint. The TXT RR provides additional information.

The <Domain> part of the service name is identical to the global (DNS subdomain) part of the authority in URIs that identify servers or groups of servers.

The <ServiceType> part is composed of at least two labels. The first label of the pair is the application protocol name [RFC6335] preceded by an underscore character. The second label indicates the transport and is always "udp" for UDP-based CoAP services. In cases where narrowing the scope of the search may be useful, these labels may be optionally preceded by a subtype name followed by the "_sub" label.
An example of this more specific <ServiceType> is "lamp._sub._dali._udp".

The default <Instance> part of the service name may be set at the factory or during the commissioning process. It SHOULD uniquely identify an instance of <ServiceType> within a <Domain>. Taken together, these three elements comprise a unique name for an SRV/ TXT record pair within the DNS subdomain.

The granularity of a service name MAY be that of a host or group, or it could represent a particular resource within a CoAP server. The SRV record contains the host name (AAAA record name) and port of the service while protocol is part of the service name. In the case where a service name identifies a particular resource, the path part of the URI must be carried in a corresponding TXT record.

A DNS TXT record is in practice limited to a few hundred octets in length, which is indicated in the resource record header in the DNS response message. The data consists of one or more strings comprising a key=value pair. By convention, the first pair is txtver=<number> (to support different versions of a service description).

9.2. mapping ins to <Instance>

The Resource Instance "ins" attribute maps to the <Instance> part of a DNS-SD service name. It is stored directly in the DNS as a single DNS label of canonical precomposed UTF-8 [RFC3629] "Net-Unicode" (Unicode Normalization Form C) [RFC5198] text. However, to the extent that the "ins" attribute may be chosen to match the DNS host name of a service, it SHOULD use the syntax defined in Section 3.5 of [RFC1034] and Section 2.1 of [RFC1123].

The <Instance> part of the name of a service being offered on the network SHOULD be configurable by the user setting up the service, so that he or she may give it an informative name. However, the device or service SHOULD NOT require the user to configure a name before it can be used. A sensible choice of default name can allow the device or service to be accessed in many cases without any manual configuration at all. The default name should be short and descriptive, and MAY include a collision-resistant substring such as the lower bits of the device’s MAC address, serial number, fingerprint, or other identifier in an attempt to make the name relatively unique.

DNS labels are currently limited to 63 octets in length and the entire service name may not exceed 255 octets.
9.3. Mapping rt to <ServiceType>

The resource type "rt" attribute is mapped into the <ServiceType> part of a DNS-SD service name and SHOULD conform to the reg-rel-type production of the Link Format defined in Section 2 of [RFC6690]. The "rt" attribute MUST be composed of at least a single Net-Unicode text string, without underscore '_' or period '.' and limited to 15 octets in length, which represents the application protocol name. This string is mapped to the DNS-SD <ServiceType> by prepending an underscore and appending a period followed by the "_udp" label. For example, rt="dali" is mapped into "_dali._udp".

The application protocol name may be optionally followed by a period and a service subtype name consisting of a Net-Unicode text string, without underscore or period and limited to 63 octets. This string is mapped to the DNS-SD <ServiceType> by prepending a period followed by the "_sub" label and then appending a period followed by the service type label pair derived as in the previous paragraph. For example, rt="dali.light" is mapped into "light._sub._dali._udp".

The resulting string is used to form labels for DNS-SD records which are stored directly in the DNS.

9.4. Domain mapping

DNS domains may be derived from the "d" attribute. The domain attribute may be suffixed with the zone name of the authoritative DNS server to generate the domain name. The "ep" attribute is prefixed to the domain name to generate the FQDN to be stored into DNS with an AAAA RR.

9.5. TXT Record key=value strings

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

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

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

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

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

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

<- 2.05 Content "<coap://[FDFD::1234]:5683/light/1>;exp;     |
     rt="dali.light";ins="Spot";                        |
     d="office";ep="node1"     

Req: GET /rd-lookup/res?exp

Res: 2.05 Content
<coap://[FDFD::1234]:5683/light/1>
    exp;rt="dali.light";ins="Spot";
    d="office";ep="node1"
```

The agent subsequently registers the following DNS-SD RRs, assuming a zone name "example.com" prefixed with "office":

```
nodel.office.example.com.          IN AAAA      FDFD::1234
_dali._udp.office.example.com    IN PTR
    Spot._dali._udp.office.example.com
light._sub._dali._udp.example.com IN PTR
    Spot._dali._udp.office.example.com
Spot._dali._udp.office.example.com IN SRV  0 0 5683
    nodel.office.example.com.
Spot._dali._udp.office.example.com IN TXT
txtver=1;path=/light/1
```

In the above figure the Service Name is chosen as Spot._dali._udp.office.example.com without the light._sub service prefix. An alternative Service Name would be: Spot.light._sub._dali._udp.office.example.com.
10. Security Considerations

The security considerations as described in Section 7 of [RFC5988] and Section 6 of [RFC6690] apply. The ".well-known/core" resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252]. DTLS or TLS based security SHOULD be used on all resource directory interfaces defined in this document.

10.1. Endpoint Identification and Authentication

An Endpoint is determined to be unique by an RD by the Endpoint identifier parameter included during Registration, and any associated TLS or DTLS security bindings. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint or Client on a resource directory SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security. Endpoints using a Certificate MUST include the Endpoint identifier as the Subject of the Certificate, and this identifier MUST be checked by a resource directory to match the Endpoint identifier included in the Registration message.

10.2. Access Control

Access control SHOULD be performed separately for the RD Function Set and the RD Lookup Function Set, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the domain, endpoint or resource level.

10.3. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly become part of a DDoS attack as UDP does not require return routability check. Therefore, an attacker can easily spoof the source IP of the target entity and send requests to such a service which would then respond to the target entity. This can be used for large-scale DDoS attacks on the target. Especially, if the service returns a response that is order of magnitudes larger than the request, the situation becomes even worse as now the attack can be amplified. DNS servers have been widely used for DDoS amplification attacks. Recently, it has been observed that NTP Servers, that also run on unprotected UDP have been used for DDoS attacks (http://tools.cisco.com/security/center/content/CiscoSecurityNotice/
CVE-2013-5211) since there is no return routability check and can have a large amplification factor. The responses from the NTP server were found to be 19 times larger than the request. A Resource Directory (RD) which responds to wild-card lookups is potentially vulnerable if run with CoAP over UDP. Since there is no return routability check and the responses can be significantly larger than requests, RDs can unknowingly become part of a DDoS amplification attack. Therefore, it is RECOMMENDED that implementations ensure return routability. This can be done, for example by responding to wild card lookups only over DTLS or TLS or TCP.

11. IANA Considerations

11.1. Resource Types

"core.rd", "core.rd-group" and "core.rd-lookup" resource types need to be registered with the resource type registry defined by [RFC6690].

11.2. Link Extension

The "exp" attribute needs to be registered when a future Web Linking link-extension registry is created (e.g. in RFC5988bis).

11.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include the human readable name of the parameter, the query parameter, validity requirements if any and a description. The query parameter MUST be a valid URI query key [RFC3986].

Initial entries in this sub-registry are as follows:
Table 1: RD Parameters

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

12. Examples

Examples are added here.

12.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the Resource Directory (RD) to facilitate the installation and start up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address. No conclusions must be drawn on the realization of actual installation procedures, because the example "emphasizes" some of the issues that may influence the use of the RD.

12.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the
installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one end-point. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager. Following the lay-out of cables and routers the network manager has defined DNS domains. The presence sensor and luminary1 are part of DNS domain: rtr_5612_rrt.example.com and luminary2 is part of rtr_7899_pfa.example.com. The names of luminary1- luminary2-, and sensor- interfaces are respectively: lm_12-345-678, lm_12-456-378, and sn_12-345-781. These names are stored in DNS together with their IP addresses. The FQDN of the interfaces is shown in Table 2 below:

<table>
<thead>
<tr>
<th>Name</th>
<th>FQDN</th>
</tr>
</thead>
<tbody>
<tr>
<td>luminary1</td>
<td>lm_12-345-678.rtr_5612_rrt.example.com</td>
</tr>
<tr>
<td>luminary2</td>
<td>lm_12-456-378.rtr_7899_pfa.example.com</td>
</tr>
<tr>
<td>Presence sensor</td>
<td>sn_12-345-781.rtr_5612_rrt.example.com</td>
</tr>
<tr>
<td>Resource directory</td>
<td>pc_123456.rtr_5612_rrt.example.com</td>
</tr>
</tbody>
</table>

Table 2: interface FQDNs

At the moment of installation, the network under installation is not necessarily connected to the DNS infra structure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and sensor shown in Table 3 below:
Table 3: interface SLAAC addresses

In Section 12.1.2 the use of resource directory during installation is presented. In Section 12.1.3 the connection to DNS is discussed.

12.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have rt: light, and the presence sensor has rt: p-sensor. The end-points have names which are relevant to the light installation manager. In this case luminary1, luminary2, and the presence sensor are located in room 2-4-015, where luminary1 is located at the window and luminary2 and the presence sensor are located at the door. The end-point names reflect this physical location. The middle, left and right lamps are accessed via path /light/middle, /light/left, and /light/right respectively. The identifiers relevant to the Resource Directory are shown in Table 4 below:

<table>
<thead>
<tr>
<th>Name</th>
<th>end-point</th>
<th>resource path</th>
<th>resource type</th>
</tr>
</thead>
<tbody>
<tr>
<td>luminary1</td>
<td>lm_R2-4-015_wndw</td>
<td>/light/left</td>
<td>light</td>
</tr>
<tr>
<td>luminary1</td>
<td>lm_R2-4-015_wndw</td>
<td>/light/middle</td>
<td>light</td>
</tr>
<tr>
<td>luminary1</td>
<td>lm_R2-4-015_wndw</td>
<td>/light/right</td>
<td>light</td>
</tr>
<tr>
<td>luminary2</td>
<td>lm_R2-4-015_door</td>
<td>/light/left</td>
<td>light</td>
</tr>
<tr>
<td>luminary2</td>
<td>lm_R2-4-015_door</td>
<td>/light/middle</td>
<td>light</td>
</tr>
<tr>
<td>luminary2</td>
<td>lm_R2-4-015_door</td>
<td>/light/right</td>
<td>light</td>
</tr>
<tr>
<td>Presence sensor</td>
<td>ps_R2-4-015_door</td>
<td>/ps</td>
<td>p-sensor</td>
</tr>
</tbody>
</table>

Table 4: Resource Directory identifiers
The CT inserts the end-points of the luminaries and the sensor in the RD using the Context parameter (con) to specify the interface address:

Req: POST coap://[FDFD::ABCD:0]/rd
    ep=lm_R2-4-015_wndw&con=coap://[FDFD::ABCD:1]
Payload:
  </light/left>;rt="light";
    d="R2-4-015";ins="lamp4444";exp,
  </light/middle>;rt="light";
    d="R2-4-015";ins="lamp5555";exp,
  </light/right>;rt="light";
    d="R2-4-015";ins="lamp6666";exp

Res: 2.01 Created
Location: /rd/4521

Req: POST coap://[FDFD::ABCD:0]/rd
    ep=lm_R2-4-015_door&con=coap://[FDFD::ABCD:2]
Payload:
  </light/left>;rt="light";
    d="R2-4-015";ins="lamp1111";exp,
  </light/middle>;rt="light";
    d="R2-4-015";ins="lamp2222";exp,
  </light/right>;rt="light";
    d="R2-4-015";ins="lamp3333";exp

Res: 2.01 Created
Location: /rd/4522

Req: POST coap://[FDFD::ABCD:0]/rd
    ep=ps_R2-4-015_door&con=coap://[FDFD::ABCD:3]
Payload:
  </ps>;rt="p-sensor";
    d="R2-4-015";ins="pres1234";exp

Res: 2.01 Created
Location: /rd/4523

The domain name d="R2-4-015" has been added for an efficient lookup because filtering on "ep" name is awkward. The same domain name is communicated to the two luminaries and the presence sensor by the CT. The "exp" attribute is set for the later administration in DNS of the instance name ins="lampxxxx".

Once the individual endpoints are registered, the group needs to be registered. Because the presence sensor sends one multicast message to the luminaries, all lamps in the group need to have an identical
This path is created on the two luminaries using the batch command defined in [I-D.ietf-core-interfaces]. The path to a batch of lamps is defined as: /light/grp1. In the example below, two endpoints are updated with an additional resource using the path /light/grp1 on the two luminaries.

Req: POST
coop://[FDFD::ABCD:1]/light/grp1
     (content-type:application/link-format)<light/middle>,<light/left>

Res: 2.04 Changed

Req: POST
coop://[FDFD::ABCD:2]/light/grp1
     (content-type:application/link-format)<light/right>

Res: 2.04 Changed

The group is specified in the RD. The Context parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, these two end-points and the end-point of the presence sensor are registered as members of the group.

It is expected that Standards Developing Organizations (SDOs) may develop other special purpose protocols to specify additional group links, group membership, group names and other parameters in the individual nodes.

Req: POST coap://[FDFD::ABCD:0]/rd-group
     ?gp=grp_R2-4-015;con="coap://[FF05::1]";exp;ins="grp1234"
Payload:
<>ep=lm_R2-4-015_wndw,
<>ep=lm_R2-4-015_door,
<>ep=ps_R2-4-015_door

Res: 2.01 Created
Location: /rd-group/501

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its domain, queries the RD for the end-point with rt=light and d=R2-4-015. The RD returns all end-points in the domain.
Knowing its own IPv6 address, the luminary discovers its endpoint name. With the end-point name the luminary queries the RD for all groups to which the end-point belongs.

Req: GET coap://[FDFD::ABCD:0]/rd-lookup/gp
     ?ep=lm_R2-4-015_wndw

Res: 2.05 Content
<coap://[FF05::1]>;gp="grp_R2-4-015"

From the context parameter value, the luminary learns the multicast address of the multicast group.

Alternatively, the CT can communicate the multicast address directly to the luminaries by using the "coap-group" resource specified in [RFC7390].

Req: POST //[FDFD::ABCD:1]/coap-group
     Content-Format: application/coap-group+json
     { "a": "[FF05::1]" }
     { "n": "grp_R2-4-015" }

Res: 2.01 Created
Location-Path: /coap-group/1

Dependent on the situation only the address ,"a", or the name, "n", is specified in the coap-group resource. Instead of the RD group name also the DNS group name can be used.

12.1.3. DNS entries

The network manager assigns the domain bc.example.com to the entries coming from the RD. The agent that looks up the resource directory uses the domain name bc.example.com as prescribed, to enter the services and hosts into the DNS.

The agent does a lookup as specified in Section 9.6. The RD returns all entries annotated with "exp". The agent subsequently registers the following DNS-SD RRs:
To ask for all lamps is equivalent to returning all PTR RR with label
_light.udp.bc.example.com. from the DNS. When it is required to
filter on the rd=R2-4-015 value in the DNS, additional PTR RRs have to be entered into the DNS.

R2-4-015._light._udp.bc.example.com IN PTR
lamp1111._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com IN PTR
lamp2222._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com IN PTR
lamp3333._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com IN PTR
lamp4444._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com IN PTR
lamp5555._light._udp.bc.example.com
R2-4-015._light._udp.bc.example.com IN PTR
lamp6666._light._udp.bc.example.com

Returning all PTR RRs with label R2-4-015._light._udp.bc.example.com provides all service instances within the domain R2-4-015. This filtering can be handy when there are many rooms. In the example there is only one room, making the filtering superfluous.

The agent can also discover groups that need to be discovered. It queries RD to return all groups which are exported.

Req: GET /rd-lookup/gp?exp

Res: 2.05 Content
<coap://[FF05::1]/>;exp;gp="grp_R2-4-015;ins="grp1234";
ep="lm_R2-4-015_wndw";
ep="lm_R2-4-015_door"

The group with FQDN grp_R2-4-015.bc.example.com can be entered into the DNS by the agent. The accompanying instance name is grp1234. The <ServiceType> is chosen to be _group._udp. The agent enters the following RRs into the DNS.

grp_R2-4-015.bc.example.com. IN AAAA FF05::1
_group._udp.bc.example.com IN PTR
grp1234._group._udp.bc.example.com
grp1234._group._udp.bc.example.com IN SRV 0 0 5683
grp_R2-4-015_door.bc.example.com.
grp1234._group._udp.bc.example.com IN TXT
ttxtver=1;path=/light/grp1
12.1.4. RD Operation

The specification of the group can be used by devices other than the luminaries and the sensor to learn the multicast address of the group in a given room. For example a smart phone may be used to adjust the lamps in the room.

After entry into the room, on request of the user, the smart phone queries the presence of RDs and may display all the domain names found on the RDs. The user can, for example, scroll all domains (room names in this case) and select the room that he entered. After selection the phone shows all groups in the selected room with their members. Selecting a group, the user can dim, switch on/off the group of lights, or possibly even create temporary new groups.

In all examples the SLAAC IPv6 address can be exchanged with the FQDN, when a connection to DNS exists. Using the FQDN, a node learns the interface’s IPv6 address, or the group’s multicast address from DNS. In the same way the presence sensor can learn the multicast address to which it should send its presence messages.

12.2. OMA Lightweight M2M (LWM2M) Example

This example shows how the OMA LWM2M specification makes use of Resource Directory (RD).

OMA LWM2M is a profile for device services based on CoAP, CoRE RD, and other IETF RFCs and drafts. LWM2M defines a simple object model and a number of abstract interfaces and operations for device management and device service enablement.

An LWM2M server is an instance of an LWM2M middleware service layer, containing a Resource Directory along with other LWM2M interfaces defined by the LWM2M specification.

CoRE Resource Directory (RD) is used to provide the LWM2M Registration interface.

LWM2M does not provide for registration domains and does not currently use the rd-group or rd-lookup interfaces.

The LWM2M specification describes a set of interfaces and a resource model used between a LWM2M device and an LWM2M server. Other interfaces, proxies, applications, and function sets are currently out of scope for LWM2M.

The location of the LWM2M Server and RD Function Set is provided by the LWM2M Bootstrap process, so no dynamic discovery of the RD.
function set is used. LWM2M Servers and endpoints are not required to implement the ./well-known/core resource.

12.2.1. The LWM2M Object Model

The OMA LWM2M object model is based on a simple 2 level class hierarchy consisting of Objects and Resources.

An LWM2M Resource is a REST endpoint, allowed to be a single value or an array of values of the same data type.

An LWM2M Object is a resource template and container type that encapsulates a set of related resources. An LWM2M Object represents a specific type of information source; for example, there is a LWM2M Device Management object that represents a network connection, containing resources that represent individual properties like radio signal strength.

Since there may potentially be more than one of a given type object, for example more than one network connection, LWM2M defines instances of objects that contain the resources that represent a specific physical thing.

The URI template for LWM2M consists of a base URI followed by Object, Instance, and Resource IDs:

{/{base-uri}} {/{object-id}} {/{object-instance}} {/{resource-id}} {/{resource-instance}}

The five variables given here are strings. base-uri can also have the special value "undefined" (sometimes called "null" in RFC 6570). Each of the variables object-instance, resource-id, and resource-instance can be the special value "undefined" only if the values behind it in this sequence also are "undefined". As a special case, object-instance can be "empty" (which is different from "undefined") if resource-id is not "undefined". [__TEMPLATE_TODO__]

base-uri := Base URI for LWM2M resources or "undefined" for default (empty) base URI

object-id := OMNA registered object ID (0-65535)

object-instance := Object instance identifier (0-65535) or "undefined"/"empty" (see above) to refer to all instances of an object ID

resource-id := OMNA registered resource ID (0-65535) or "undefined" to refer to all resources within an instance
resource-instance := Resource instance identifier or "undefined" to refer to single instance of a resource

LWM2M IDs are 16 bit unsigned integers represented in decimal (no leading zeroes except for the value 0) by URI format strings. For example, a LWM2M URI might be:

/1/0/1

The base uri is empty, the Object ID is 1, the instance ID is 0, the resource ID is 1, and the resource instance is "undefined". This example URI points to internal resource 1, which represents the registration lifetime configured, in instance 0 of a type 1 object (LWM2M Server Object).

12.2.2. LWM2M Register Endpoint

LWM2M defines a registration interface based on the Resource Directory Function Set, described in Section 5. The URI of the LWM2M Resource Directory function set is specified to be "/rd" as recommended in Section 5.2.

LWM2M endpoints register object IDs, for example </1>, to indicate that a particular object type is supported, and register object instances, for example </1/0>, to indicate that a particular instance of that object type exists.

Resources within the LWM2M object instance are not registered with the RD, but may be discovered by reading the resource links from the object instance using GET with a CoAP Content-Format of application/link-format. Resources may also be read as a structured object by performing a GET to the object instance with a Content-Format of senml+json.

When an LWM2M object or instance is registered, this indicates to the LWM2M server that the object and it’s resources are available for management and service enablement (REST API) operations.

LWM2M endpoints may use the following RD registration parameters as defined in Table 1 :

ep - Endpoint Name
lt - registration lifetime

Endpoint Name is mandatory, all other registration parameters are optional.

Additional optional LWM2M registration parameters are defined:
Table 5: LWM2M Additional Registration Parameters

The following RD registration parameters are not currently specified for use in LWM2M:

et - Endpoint Type
con - Context

The endpoint registration must include a payload containing links to all supported objects and existing object instances, optionally including the appropriate link-format relations.

Here is an example LWM2M registration payload:

</1>,</1/0>,</3/0>,</5>

This link format payload indicates that object ID 1 (LWM2M Server Object) is supported, with a single instance 0 existing, object ID 3 (LWM2M Device object) is supported, with a single instance 0 existing, and object 5 (LWM2M Firmware Object) is supported, with no existing instances.

12.2.3.  Alternate Base URI

If the LWM2M endpoint exposes objects at a base URI other than the default empty base path, the endpoint must register the base URI using rt="oma.lwm2m". An example link payload using alternate base URI would be:

</my_lwm2m>;rt="oma.lwm2m",</my_lwm2m/1>,<my_lwm2m/1/0>,<my_lwm2m/5>

This link payload indicates that the lwm2m objects will be placed under the base URI "/my_lwm2m" and that object ID 1 (server) is supported, with a single instance 0 existing, and object 5 (firmware update) is supported.
12.2.4. LWM2M Update Endpoint Registration

An LWM2M Registration update proceeds as described in Section 5.3, and adds some optional parameter updates:

lt - Registration Lifetime
b - Protocol Binding
sms - MSISDN
link payload - new or modified links

A Registration update is also specified to be used to update the LWM2M server whenever the endpoint’s UDP port or IP address are changed.

12.2.5. LWM2M De-Register Endpoint

LWM2M allows for de-registration using the delete method on the returned location from the initial registration operation. LWM2M de-registration proceeds as described in Section 5.4.

13. Acknowledgments

Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Mohit Sethi, Sampo Ukkola and Linyi Tian have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

14. Changelog

Changes from -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
o Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

o Added a catalogue use case.

o Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.

o Additional examples section added for more complex use cases.

o New DNS-SD mapping section.

o Added text on endpoint identification and authentication.

o Error code 4.04 added to Registration Update and Delete requests.

o Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

o Removed the ETag validation feature.

o Place holder for the DNS-SD mapping section.

o Explicitly disabled GET or POST on returned Location.

o New registry for RD parameters.

o Added support for the JSON Link Format.

o Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

o Updated the version and date.

Changes from -04 to -05:

o Restricted Update to parameter updates.

o Added pagination support for the Lookup interface.

o Minor editing, bug fixes and reference updates.

o Added group support.
- Changed rt to et for the registration and update interface.

Changes from -03 to -04:
- Added the ins= parameter back for the DNS-SD mapping.
- Integrated the Simple Directory Discovery from Carsten.
- Editorial improvements.
- Fixed the use of ETags.
- Fixed tickets 383 and 372

Changes from -02 to -03:
- Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- Updated REST interface descriptions to use RFC6570 URI Template format.
- Introduced an improved RD Lookup design as its own function set.
- Improved the security considerations section.
- Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:
- Added a terminology section.
- Changed the inclusion of an ETag in registration or update to a MAY.
- Added the concept of an RD Domain and a registration parameter for it.
- Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.
15. References

15.1. Normative References

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Resource Identifier (URI): Generic Syntax", STD 66, RFC
3986, January 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an
IANA Considerations Section in RFCs", BCP 26, RFC 5226,
May 2008.


Cheshire, "Internet Assigned Numbers Authority (IANA)
Procedures for the Management of the Service Name and
Transport Protocol Port Number Registry", BCP 165, RFC
6335, August 2011.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M.,

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

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service
Discovery", RFC 6763, February 2013.

15.2. Informative References

[I-D.ietf-core-interfaces]


[RFC1123] Braden, R., "Requirements for Internet Hosts - Application


Editorial Comments

[_TEMPLATE_TODO] This text needs some help from an RFC 6570 expert.

Authors’ Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

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

Michael Koster
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-576-1500 x11516
Email: Michael.Koster@arm.com
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.
Connecting sensors to the internet is not new, and there have been many protocols designed to facilitate it. This specification defines new media types for carrying simple sensor information in a protocol.
such as HTTP or CoAP [RFC7252] called the Sensor Markup Language (SenML). This format was designed so that processors with very limited capabilities could easily encode a sensor measurement into the media type, while at the same time a server parsing the data could relatively efficiently collect a large number of sensor measurements. There are many types of more complex measurements and measurements that this media type would not be suitable for. A decision was made not to carry most of the meta data about the sensor in this media type to help reduce the size of the data and improve efficiency in decoding. Instead meta-data about a sensor resource can be described out-of-band using the CoRE Link Format [RFC6690]. The markup language can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and the web resource model where the sensor is requested as a resource representation (GET /sensor/temperature).

SenML is defined by a data model for measurements and simple meta-data about measurements and devices. The data is structured as a single object (with attributes) that contains an array of entries. Each entry is an object that has attributes such as a unique identifier for the sensor, the time the measurement was made, and the current value. Serializations for this data model are defined for JSON [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.

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

In the example above, the array in the object has a single measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a temperature of 23.5 degrees Celsius.

2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets on mesh networks from large numbers of constrained devices. Keeping the total size of payload under 80 bytes makes this easy to use on a wireless mesh network. It is always difficult to define what small code is, but there is a desire to be able to implement this in roughly 1 KB of flash on a 8 bit microprocessor. Experience with Google power meter and large scale deployments has indicated that the solution needs to support allowing multiple measurements to be batched into a single HTTP or CoAP request. This "batch" upload capability allows the server side to efficiently support a large number of devices. It also conveniently supports batch transfers from proxies and storage devices, even in situations
where the sensor itself sends just a single data item at a time. The multiple measurements could be from multiple related sensors or from the same sensor but at different times.

3. Terminology

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

4. Semantics

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

Base Name

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

Base Time

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

Base Units

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

Version

Version number of media type format. This attribute is optional positive integer and defaults to 1 if not present.

Measurement or Parameter Entries

Array of values for sensor measurements or other generic parameters (such as configuration parameters). If present there must be at least one entry in the array.
Each array entry contains several attributes, some of which are optional and some of which are mandatory.

Name

Name of the sensor or parameter. When appended to the Base Name attribute, this must result in a globally unique identifier for the resource. The name is optional, if the Base Name is present. If the name is missing, Base Name must uniquely identify the resource. This can be used to represent a large array of measurements from the same sensor without having to repeat its identifier on every measurement.

Units

Units for a measurement value.

Value

Value of the entry. Optional if a Sum value is present, otherwise required. Values are represented using three basic data types, Floating point numbers ("v" field for "Value"), Booleans ("bv" for "Boolean Value") and Strings ("sv" for "String Value"). Exactly one of these three fields MUST appear.

Sum

Integrated sum of the values over time. Optional. This attribute is in the units specified in the Unit value multiplied by seconds.

Time

Time when value was recorded. Optional.

Update Time

A time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. This can be used to detect the failure of sensors or communications path from the sensor. Optional.
The SenML format can be extended with further custom attributes placed in the base object, or in an entry. Extensions in the base object pertain to all entries, whereas extensions in an entry object only pertain to that.

Systems reading one of the objects MUST check for the Version attribute. If this value is a version number larger than the version which the system understands, the system SHOULD NOT use this object. This allows the version number to indicate that the object contains mandatory to understand attributes. New version numbers can only be defined in an RFC that updates this specification or its successors.

The Name value is concatenated to the Base Name value to get the name of the sensor. The resulting name needs to uniquely identify and differentiate the sensor from all others. If the object is a representation resulting from the request of a URI [RFC3986], then in the absence of the Base Name attribute, this URI is used as the default value of Base Name. Thus in this case the Name field needs to be unique for the URI, for example an index or subresource name of sensors handled by the URI.

Alternatively, for objects not related to a URI, a unique name is required. In any case, it is RECOMMENDED that the full names are represented as URIs or URNs [RFC2141]. One way to create a unique name is to include a EUI-48 or EUI-64 identifier (A MAC address) or some other bit string that is guaranteed uniqueness (such as a 1-wire address) that is assigned to the device. Some of the examples in this draft use the device URN type as specified in [I-D.arkko-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name.

The resulting concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", ",", ",", "\", or "_" and it MUST start with a character out of the set "A" to "Z", "a" to "z", or "0" to "9". This restricted character set was chosen so that these names can be directly used as in other types of URI including segments of an HTTP path with no special encoding. [RFC5952] contains advice on encoding an IPv6 address in a name.

If either the Base Time or Time value is missing, the missing attribute is considered to have a value of zero. The Base Time and Time values are added together to get the time of measurement. A time of zero indicates that the sensor does not know the absolute time and the measurement was made roughly "now". A negative value is used to indicate seconds in the past from roughly "now". A positive value is used to indicate the number of seconds, excluding leap seconds, since the start of the year 1970 in UTC.
Representing the statistical characteristics of measurements can be very complex. Future specification may add new attributes to provide better information about the statistical properties of the measurement.

5. Associating Meta-data

SenML is designed to carry the minimum dynamic information about measurements, and for efficiency reasons does not carry more static meta-data about the device, object or sensors. Instead, it is assumed that this meta-data is carried out of band. For web resources using SenML representations, this meta-data can be made available using the CoRE Link Format [RFC6690].

The CoRE Link Format provides a simple way to describe Web Links, and in particular allows a web server to describe resources it is hosting. The list of links that a web server has available, can be discovered by retrieving the /.well-known/core resource, which returns the list of links in the CoRE Link Format. Each link may contain attributes, for example title, resource type, interface description and content-type.

The most obvious use of this link format is to describe that a resource is available in a SenML format in the first place. The relevant media type indicator is included in the Content-Type (ct=) attribute.

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

6. JSON Representation (application/senml+json)

Root variables:
It is RECOMMENDED that in textual JSON format, when present, the attributes appear in the above order. However, implementations MUST be able to process them in any order.

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

The root contents MUST consist of exactly one JSON object as specified by [RFC7159]. This object MAY contain a "bn" attribute with a value of type string. This object MAY contain a "bt" attribute with a value of type number. The object MAY contain a "bu" attribute with a value of type string. The object MAY contain a "ver" attribute with a value of type number. The object MAY contain other attribute value pairs, and the object MUST contain exactly one "e" attribute with a value of type array. The array MUST have one or more measurement or parameter objects.

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

Systems receiving measurements MUST be able to process the range of floating point numbers that are representable as an IEEE double-precision floating-point numbers [IEEE.754.1985]. The number of significant digits in any measurement is not relevant, so a reading of 1.1 has exactly the same semantic meaning as 1.10. If the value has an exponent, the "e" MUST be in lower case. The mantissa SHOULD be less than 19 characters long and the exponent SHOULD be less than 5 characters long. This allows time values to have better than microsecond precision over the next 100 years.

6.1. Examples

6.1.1. Single Datapoint

The following shows a temperature reading taken approximately "now" by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

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

6.1.2. Multiple Datapoints

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

{"bn": "urn:dev:mac:0024befffe804ff1/",
 "e": [
   { "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.
Note that in some usage scenarios of SenML the implementations MAY store or transmit SenML in a stream-like fashion, where data is collected over time and continuously added to the object. This mode of operation is optional, but systems or protocols using SenML in this fashion MUST specify that they are doing this. In this situation the SenML stream can be sent and received in a partial fashion, i.e., a measurement entry can be read as soon as it is received and only not when the entire SenML object is complete.

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

```json
{"bn": "http://[2001:db8::1]",
"bt": 1320067464,
"bu": "%RH",
"e":[
  { "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.

```json
{"bn": "http://[2001:db8::1]",
"bt": 1320067464,
"bu": "%RH",
"e":
[  {
    "v": 20.0, "t": 0 
  },
  {"sv": "E 24’ 30.621", "u": "lon", "t": 0 },
  {"sv": "N 60’ 7.965", "u": "lat", "t": 0 },
  {"v": 20.3, "t": 60 },
  {"sv": "E 24’ 30.622", "u": "lon", "t": 60 },
  {"sv": "N 60’ 7.965", "u": "lat", "t": 60 },
  {"v": 20.7, "t": 120 },
  {"sv": "E 24’ 30.623", "u": "lon", "t": 120 },
  {"sv": "N 60’ 7.966", "u": "lat", "t": 120 },
  {"v": 98.0, "u": "%EL", "t": 150 },
  {"v": 21.2, "t": 180 },
  {"sv": "E 24’ 30.628", "u": "lon", "t": 180 },
  {"sv": "N 60’ 7.967", "u": "lat", "t": 180 }]
}
```

6.1.4. Collection of Resources

The following example shows how to query one device that can provide multiple measurements. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on http://[2001:db8::2] on Mon Oct 31 16:27:09 UTC 2011, and has gotten two separate values as a result, a temperature and humidity measurement.

```json
{"bn": "http://[2001:db8::2]/",
"bt": 1320078429,
"ver": 1,
"e":
[  {
    "n": "temperature", "v": 27.2, "u": "Cels" },
  {"n": "humidity", "v": 80, "u": "%RH" }]
}
```

7. CBOR Representation (application/senml+cbor)

The CBOR [RFC7049] representation is equivalent to the JSON representation, with the following changes:

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

For JSON Numbers, the CBOR representation can use integers, floating point numbers, or decimal fractions (CBOR Tag 4); the common limitations of JSON implementations are not relevant for these. For the version number, however, only an unsigned integer is allowed.

<table>
<thead>
<tr>
<th>Name</th>
<th>JSON label</th>
<th>CBOR label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>ver</td>
<td>-1</td>
</tr>
<tr>
<td>Measurement or Parameters</td>
<td>e</td>
<td>-2</td>
</tr>
<tr>
<td>Base Name</td>
<td>bn</td>
<td>-3</td>
</tr>
<tr>
<td>Base Time</td>
<td>bt</td>
<td>-4</td>
</tr>
<tr>
<td>Base Units</td>
<td>bu</td>
<td>-5</td>
</tr>
<tr>
<td>Name</td>
<td>n</td>
<td>0</td>
</tr>
<tr>
<td>Units</td>
<td>u</td>
<td>1</td>
</tr>
<tr>
<td>Value</td>
<td>v</td>
<td>2</td>
</tr>
<tr>
<td>String Value</td>
<td>sv</td>
<td>3</td>
</tr>
<tr>
<td>Boolean Value</td>
<td>bv</td>
<td>4</td>
</tr>
<tr>
<td>Value Sum</td>
<td>s</td>
<td>5</td>
</tr>
<tr>
<td>Time</td>
<td>t</td>
<td>6</td>
</tr>
<tr>
<td>Update Time</td>
<td>ut</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure 1: CBOR representation: integers for map keys

8. XML Representation (application/senml+xml)

A SenML object can also be represented in XML format as defined in this section. The following example shows an XML example for the same sensor measurement as in Section 6.1.2.
<?xml version="1.0" encoding="UTF-8"?>
<senml xmlns="urn:ietf:params:xml:ns:senml"
bn="urn:dev:mac:0024befffe804ff1/
bt="1276020076"
ver="1" bu="A">
  <e n="voltage" u="V" v="120.1" />
  <e n="current" t="-5" v="1.2" />
  <e n="current" t="-4" v="1.30" />
  <e n="current" t="-3" v="0.14e1" />
  <e n="current" t="-2" v="1.5" />
  <e n="current" t="-1" v="1.6" />
  <e n="current" t="0" v="1.7" />
</senml>

The RelaxNG schema for the XML is:

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

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

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

start = senml

9. EXI Representation (application/senml-exi)

For efficient transmission of SenML over e.g. a constrained network, Efficient XML Interchange (EXI) can be used. This encodes the XML Schema structure of SenML into binary tags and values rather than ASCII text. An EXI representation of SenML SHOULD be made using the
strict schema-mode of EXI. This mode however does not allow tag extensions to the schema, and therefore any extensions will be lost in the encoding. For uses where extensions need to be preserved in EXI, the non-strict schema mode of EXI MAY be used.

The EXI header option MUST be included. An EXI schemaID options MUST be set to the value of "a" indicating the scheme provided in this specification. Future revisions to the schema can change this schemaID to allow for backwards compatibility. When the data will be transported over CoAP or HTTP, an EXI Cookie SHOULD NOT be used as it simply makes things larger and is redundant to information provided in the Content-Type header.

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="urn:ietf:params:xml:ns:senml"
  xmlns:ns1="urn:ietf:params:xml:ns:senml">
  <xs:element name="e">
    <xs:complexType>
      <xs:attribute name="n" type="xs:string"/>
      <xs:attribute name="u" type="xs:string"/>
      <xs:attribute name="v" type="xs:float"/>
      <xs:attribute name="sv" type="xs:string"/>
      <xs:attribute name="bv" type="xs:boolean"/>
      <xs:attribute name="s" type="xs:decimal"/>
      <xs:attribute name="t" type="xs:int"/>
      <xs:attribute name="ut" type="xs:int"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="senml">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="ns1:e"/>
      </xs:sequence>
      <xs:attribute name="bn" type="xs:string"/>
      <xs:attribute name="bt" type="xs:int"/>
      <xs:attribute name="bu" type="xs:string"/>
      <xs:attribute name="ver" type="xs:int"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
The following shows a hexdump of the EXI produced from encoding the following XML example. Note that while this example is similar to the first example in Section 6.1.2 in JSON format.

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

Which compresses to the following displayed in hexdump:

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

The above example used the bit packed form of EXI but it is also possible to use a byte packed form of EXI which can makes it easier for a simple sensor to produce valid EXI without really implementing EXI. Consider the example of a temperature sensor that produces a value in tenths of degrees Celsius over a range of 0.0 to 55.0. It would produce XML SenML file such as:

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

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

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

A small temperature sensor devices that only generates this one EXI file does not really need an full EXI implementation. It can simple hard code the output replacing the one wire device ID starting at byte 0x14 and going to byte 0x23 with it’s device ID, and replacing the value "0xe7 0x01" at location 0x33 to 0x34 with the current temperature. The EXI Specification [W3C.REC-exi-20110310] contains the full information on how floating point numbers are represented, but for the purpose of this sensor, the temperature can be converted.
to an integer in tenths of degrees (231 in this example). EXI stores 7 bits of the integer in each byte with the top bit set to one if there are further bytes. So the first bytes at location 0x33 is set to low 7 bits of the integer temperature in tenths of degrees plus 0x80. In this example 231 & 0x7F + 0x80 = 0xE7. The second byte at location 0x34 is set to the integer temperature in tenths of degrees right shifted 7 bits. In this example 231 >> 7 = 0x01.

10. Usage Considerations

The measurements support sending both the current value of a sensor as well as the an integrated sum. For many types of measurements, the sum is more useful than the current value. For example, an electrical meter that measures the energy a given computer uses will typically want to measure the cumulative amount of energy used. This is less prone to error than reporting the power each second and trying to have something on the server side sum together all the power measurements. If the network between the sensor and the meter goes down over some period of time, when it comes back up, the cumulative sum helps reflect what happened while the network was down. A meter like this would typically report a measurement with the units set to watts, but it would put the sum of energy used in the "s" attribute of the measurement. It might optionally include the current power in the "v" attribute.

While the benefit of using the integrated sum is fairly clear for measurements like power and energy, it is less obvious for something like temperature. Reporting the sum of the temperature makes it easy to compute averages even when the individual temperature values are not reported frequently enough to compute accurate averages. Implementors are encouraged to report the cumulative sum as well as the raw value of a given sensor.

Applications that use the cumulative sum values need to understand they are very loosely defined by this specification, and depending on the particular sensor implementation may behave in unexpected ways. Applications should be able to deal with the following issues:

1. Many sensors will allow the cumulative sums to "wrap" back to zero after the value gets sufficiently large.

2. Some sensors will reset the cumulative sum back to zero when the device is reset, loses power, or is replaced with a different sensor.

3. Applications cannot make assumptions about when the device started accumulating values into the sum.
Typically applications can make some assumptions about specific sensors that will allow them to deal with these problems. A common assumption is that for sensors whose measurement values are always positive, the sum should never get smaller; so if the sum does get smaller, the application will know that one of the situations listed above has happened.

11. IANA Considerations

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

11.1. Units Registry

IANA will create a registry of unit symbols. The primary purpose of this registry is to make sure that symbols uniquely map to give type of measurement. Definitions for many of these units can be found in [NIST822] and [BIPM].

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

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

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider the following guidelines:

1. There needs to be a real and compelling use for any new unit to be added.

2. Units should define the semantic information and be chosen carefully. Implementors need to remember that the same word may be used in different real-life contexts. For example, degrees when measuring latitude have no semantic relation to degrees when measuring temperature; thus two different units are needed.
3. These measurements are produced by computers for consumption by computers. The principle is that conversion has to be easily be done when both reading and writing the media type. The value of a single canonical representation outweighs the convenience of easy human representations or loss of precision in a conversion.

4. Use of SI prefixes such as "k" before the unit is not allowed. Instead one can represent the value using scientific notation such a 1.2e3.

5. For a given type of measurement, there will only be one unit type defined. So for length, meters are defined and other lengths such as mile, foot, light year are not allowed. For most cases, the SI unit is preferred.

6. Symbol names that could be easily confused with existing common units or units combined with prefixes should be avoided. For example, selecting a unit name of "mph" to indicate something that had nothing to do with velocity would be a bad choice, as "mph" is commonly used to mean miles per hour.

7. The following should not be used because the are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, n, u, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi.

8. The following units should not be used as they are commonly used to represent other measurements Ky, Gal, dyn, etg, P, St, Mx, G, Oe, 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.

11.2. Media Type Registration

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

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification.
11.2.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC7159]. Specifically, only the ASCII [RFC0020] subset of the UTF-8 characters are allowed. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any JSON key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "ver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Additional information:

Magic number(s): none

File extension(s): senml

Macintosh file type code(s): none
11.2.2.  senml+cbor Media Type Registration

Type name: application
Subtype name: senml+cbor
Required parameters: none
Optional parameters: none
Encoding considerations: TBD
Security considerations: TBD
Interoperability considerations: TBD
Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Additional information:

Magic number(s): none
File extension(s): senml
Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON
Restrictions on usage: None
11.2.3.  senml+xml Media Type Registration

Type name: application
Subtype name: senml+xml
Required parameters: none
Optional parameters: none
Encoding considerations: TBD
Security considerations: TBD
Interoperability considerations: TBD
Published specification: RFC-AAAA
Applications that use this media type: TBD
Additional information:
Magic number(s): none
File extension(s): senml
Macintosh file type code(s): none
Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON
Restrictions on usage: None

11.2.4.  senml-exi Media Type Registration

Type name: application
Subtype name: senml-exi
Required parameters: none
Optional parameters: none
Encoding considerations: TBD
Security considerations: TBD
Interoperability considerations: TBD
Published specification: RFC-AAAA
Applications that use this media type: TBD
Additional information:
Magic number(s): none
File extension(s): senml
Macintosh file type code(s): none
Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>
Intended usage: COMMON
Restrictions on usage: None
Author: Cullen Jennings <fluffy@iii.ca>
Change controller: IESG

11.3. XML Namespace Registration

This document registers the following XML namespaces in the IETF XML registry defined in [RFC3688].
URI: urn:ietf:params:xml:ns:senml
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces
12. Security Considerations

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

13. Privacy Considerations

Sensor data can range from information with almost no security considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transport protocol such as S/MIME or HTTP with TLS that can provide integrity, confidentiality, and authentication information about the source of the data.

14. Acknowledgement

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

The CBOR Representation text was contributed by Carsten Bormann.

15. References

15.1. Normative References


15.2. Informative References


[UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics, 2013.


Authors' Addresses

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA  95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Zach Shelby
ARM
150 Rose Orchard
San Jose  95134
FINLAND

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

Jari Arkko
Ericsson
Jorvas  02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas  02420
Finland

Email: ari.keranen@ericsson.com
Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

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

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


This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format [RFC6570] is used to describe the REST interfaces defined in this specification.

This specification makes use of the following additional terminology:

Publish-Subscribe (pub/sub): A messaging paradigm where messages are published to a broker and potential receivers can subscribe to the broker to receive messages. The publishers do not (need to) know where the message will be eventually sent: the publications and subscriptions are matched by a broker and publications are delivered by the broker to subscribed receivers.

CoAP pub/sub function set: A group of well-known REST resources that together provide the CoAP pub/sub service.

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

CoAP pub/sub Client: A CoAP client that implements the CoAP pub/sub function set.
Topic: A unique identifier for a particular item being published and/or subscribed to. A broker uses the topics to match subscriptions to publications.

3. Architecture

3.1. CoAP pub/sub Architecture

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

```
Clients         pub/sub         Broker
+-------+         |         +-------+
  | CoAP |         |       | CoAP |
  | pub/sub |------|-------+   |
  +-------+         |      +----| Broker |
  ++-------+         |      +----|       |
  | CoAP |         |       |       |
  | pub/sub |------|-------+   |
  +-------+         |      +----|       |
  ++-------+         |      +----|       |
```

Figure 1: CoAP pub/sub Architecture

3.2. CoAP pub/sub Broker

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

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

3.4. CoAP pub/sub Topic

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

4. CoAP pub/sub Function Set

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

4.1. DISCOVER

CoAP pub/sub Clients discover CoAP pub/sub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [I-D.ietf-core-resource-directory]. A CoAP pub/sub Broker SHOULD indicate its presence and availability on a network by exposing a link to its pub/sub function set at its .well-known/core location [RFC6690]. A CoAP pub/sub broker MAY register its pub/sub function set location with a Resource Directory. A broker wishing to advertise the CoAP pub/sub Function Set for Simple Discovery or through a Resource Directory MUST use the link relation rt="core.ps".

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

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

URI Template Variables:

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

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

The following response codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the broker resource. A pub/sub broker SHOULD use the value "ps" for the link subject variable whenever possible.

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

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

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

4.2. CREATE

Clients create topics on the broker using the CREATE interface. A client wishing to create a topic MUST use CoAP POST to the pub/sub function set location with a payload indicating the desired topic. The topic MUST use the CoRE link format [RFC6690]. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Type) relation in the link-format payload. The client MAY indicate the lifetime of the topic by including the Max-Age option in the CREATE request. Broker MUST return a response code of "2.01 Created" if the topic is created. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if a new topic can not be created. Broker SHOULD remove topics if
the Max-Age of the topic is exceeded without any publishes to the topic.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

URI Template: /{+ps}

URI Template Variables:

ps := pub/sub Function Set path (mandatory). The path of the pub/sub Function Set, as obtained from discovery. A pub/sub broker SHOULD use the value "ps" for this variable whenever possible.

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for this interface:

Success: 2.01 "Created". Successful Creation of the topic

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.03 "Forbidden". Topic already exists.

Failure: 4.06 "Not Acceptable". Unsupported content format for topic.

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

Client                                          Broker

| ------------ POST /ps "<topic1>;ct=50" ------------|
| <---------------- 2.01 Created ------------------|

Figure 3: Example of CREATE
4.3. PUBLISH

A CoAP pub/sub Client uses the PUBLISH interface for updating topics on the broker. The client MUST use the PUT method to publish state updates to the CoAP pub/sub Broker. A client MUST use the content format specified upon creation of a given topic to publish updates to that topic. The broker MUST reject publish operations which do not use the specified content format. A CoAP client publishing on a topic MAY indicate the maximum lifetime of the value by including the Max-Age option in the publish request. A client MAY use confirmable (CON) or non-confirmable (NON) messages to publish updates to a broker. The broker MUST return a response code of "2.04 Changed" if the publish is accepted or "4.04 Not Found" if the topic does not exist. A broker MAY return "4.29 Too Many Requests" if simple flow control as described in Section 7 is implemented.

The Broker MUST notify all clients subscribed on a particular topic each time it receives a publish on that topic. An example is shown in Figure 5. If a client publishes to a broker using non-confirmable messages, the broker MAY notify subscribed clients using non-confirmable messages. If a client publishes to a broker using confirmable messages, the broker MUST also notify all subscribed clients using confirmable messages. If a client publishes to a broker with the Max-Age option, the broker MUST include the same value for the Max-Age option in all notifications. A broker MUST use CoAP Notification as described in [I-D.ietf-core-observe] to notify subscribed clients.

The PUBLISH interface is specified as follows:

Interaction: Client -> Broker

Method: PUT

URI Template: /{ps}/{topic}

URI Template Variables:

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

topic :=   The desired topic to publish on.

Content-Format:   Any valid CoAP content format

Payload:   Representation of the topic value (CoAP resource state representation) in the indicated content format
The following response codes are defined for this interface:

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

**Failure:** 4.00 "Bad Request". Malformed request.

**Failure:** 4.01 "Unauthorized". Authorization failure.

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

**Failure:** 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 7).

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

```
Client                                      Broker
  |------------------------ PUT /ps/topic1 "1033.3" ------>
  |                                                      |
  |                                                      |
  <------------------------ 2.04 Changed-------------------->
```

**Figure 4: Example of PUBLISH**

### 4.4. SUBSCRIBE

CoAP pub/sub Clients subscribe to topics on the Broker using CoAP Observe as described in [I-D.ietf-core-observe]. A CoAP pub/sub Client wishing to Subscribe to a topic on a broker MUST use a CoAP GET with Observe registration. The Broker MAY add the client to a list of observers. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. The broker MUST accept Subscribe requests on a topic if the content format of the request matches the content format the topic was created with. The broker MAY accept Subscribe requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired.
The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if it can not return the requested content format.

The SUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

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

URI Template Variables:

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

topic := The desired topic to subscribe to.

The following response codes are defined for this interface:

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

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

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.15 "Unsupported Content Format". Unsupported content format.

Figure 5 shows an example of Client2 subscribing to "topic1" and receiving a response from the broker, with a subsequent notification. The subscribe response from the broker uses the last stored value associated with the topic1. The notification from the broker is sent in response to the publish received from Client1.
4.5. UNSUBSCRIBE

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

The UNSUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

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

URI Template Variables:

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

topic := The desired topic to unsubscribe from.

The following response codes are defined for this interface:

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

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

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

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

Figure 6: Example of UNSUBSCRIBE

4.6. READ

A CoAP pub/sub client wishing to obtain only the most recent published value on a topic MAY use the READ interface. For reading, the client uses the CoAP GET method. The broker MUST accept Read requests on a topic if the content format of the request matches the content format the topic was created with. The broker MAY accept Read requests which specify content formats that the broker can supply as alternate content formats to the content format the topic was registered with. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the broker can supply the requested content format. If the topic was published with the Max-Age option, the broker MUST set the Max-Age option in the valid response to the amount of time remaining for the topic to be valid since the last publish. The Broker MUST return a response code of "2.04 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if the broker can not return the requested content format.

The READ interface is specified as follows:

Interaction: Client -> Broker

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

URI Template Variables:

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

topic := The desired topic to READ.

The following response codes are defined for this interface:

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

Success: 2.04 "No Content". Topic exists, value not included.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.15 "Unsupported Content Format". Unsupported content-format.

Figure 7 shows an example of a successful READ from topic1, followed by a Publish on the topic, followed at some time later by a read of the updated value from the recent Publish.

```
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 7: Example of READ
4.7. REMOVE

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

The REMOVE interface is specified as follows:

Interaction: Client -> Broker

Method: DELETE

URI Template: /{ps}/{topic}

URI Template Variables:

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

topic := The desired topic to REMOVE.

Content-Format: None

Response Payload: None

The following response codes are defined for this interface:

Success:  2.02 "Deleted". Successful remove
Failure:  4.00 "Bad Request". Malformed request.
Failure:  4.01 "Unauthorized". Authorization failure.
Failure:  4.04 "Not Found". Topic does not exist.

Figure 8 shows a successful remove of topic1.

A CoAP pub/sub Broker may register a pub/sub Function Set with a Resource Directory. A pub/sub Client may use an RD to discover a pub/sub Broker.

A CoAP pub/sub Client may register CoRE Links [RFC6690] to created pub/sub Topics with an RD. A pub/sub Client may use an RD to discover pub/sub Topics. A client which registers pub/sub Topics with an RD MUST use the context relation (con) [I-D.ietf-core-resource-directory] to indicate that the context of the registered links is the pub/sub Broker.

6. Sleep-Wake Operation

CoAP pub/sub provides a way for client nodes to sleep between operations, conserving energy during idle periods. This is made possible by shifting the server role to the broker, allowing the broker to be always-on and respond to requests from other clients while a particular client is sleeping.

For example, the broker will retain the last state update received from a sleeping client, in order to supply the most recent state update to other clients in response to read and subscribe operations.

Likewise, the broker will retain the last state update received on the topic such that a sleeping client, upon waking, can perform a read operation to the broker to update its own state from the most recent system state update.

7. Simple Flow Control

Since the broker node has to potentially send a large amount of notification messages for each publish message and it may be serving a large amount of subscribers and publishers simultaneously, the broker may become overwhelmed if it receives many publish messages to popular topics in a short period of time.
If the broker is unable to serve a certain client that is sending publish messages too fast, the broker MUST respond with Response Code 4.29, "Too Many Requests". This Response Code is like HTTP 429 "Too Many Requests" but uses the Max-Age Option in place of the "Retry-After" header field to indicate the number of seconds after which to retry. The broker MAY stop creating notifications from the publish messages from this client and to this topic for the indicated time.

If a client receives the 4.29 Response Code from the broker for a publish message to a topic, it MUST NOT send new publish messages to the broker on the same topic before the time indicated in Max-Age has passed.

8. Security Considerations

CoAP pub/sub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP pub/sub broker and the clients SHOULD authenticate each other and enforce access control policies. A malicious client could subscribe to data it is not authorized to or mount a denial of service attack against the broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP pub/sub broker introduces challenges for the use of end-to-end security between for example a client device on a sensor network and a client application running in a cloud-based server infrastructure since brokers terminate the exchange. While running separate DTLS sessions from the client device to the broker and from broker to client application protects confidentially on those paths, the client device does not know whether the commands coming from the broker are actually coming from the client application. Similarly, a client application requesting data does not know whether the data originated on the client device. For scenarios where end-to-end security is desirable the use of application layer security is unavoidable. Application layer security would then provide a guarantee to the client device that any request originated at the client application. Similarly, integrity protected sensor data from a client device will also provide guarantee to the client application that the data originated on the client device itself. The protected data can also be verified by the intermediate broker ensuring that it stores/caches correct request/response and no malicious messages/
requests are accepted. The broker would still be able to perform aggregation of data/requests collected.

Depending on the level of trust users and system designers place in the CoAP pub/sub broker, the use of end-to-end object security is RECOMMENDED [I-D.selander-ace-object-security].

9. IANA Considerations

This document registers one attribute value in the Resource Type (rt=) registry established with [RFC6690] and appends to the definition of one CoAP Response Code in the CoRE Parameters Registry.

9.1. Resource Type value 'core.ps'
   - Attribute Value: core.ps
   - Description: Section 4 of [[This document]]
   - Reference: [[This document]]
   - Notes: None

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

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

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

11. References

11.1. Normative References

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

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

[I-D.selander-ace-object-security]


11.2. Informative References

Authors’ Addresses

Michael Koster
ARM Limited
Email: Michael.Koster@arm.com

Ari Keranen
Ericsson
Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson
Email: jaime.jimenez@ericsson.com
Abstract

The Resource Directory (RD) is a key element for successful deployments of constrained networks. Similar to the HTTP web search engines (e.g., Google, Bing), the RD for CoAP should also support useful search query responses beyond a basic listing of relevant links. This document proposes several new features to be considered for the RD. The only goal of this document is to trigger discussion in the CORE WG so that all relevant features for RD evolution are taken into account during CORE re-charter activities.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 6, 2016.

Copyright Notice

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

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

Table of Contents

1. Terminology and Conventions ............................ 2
2. Background .................................................. 2
3. Proposal ....................................................... 3
4. Summary ......................................................... 4
5. Acknowledgements ............................................ 4
6. IANA Considerations .......................................... 4
7. Security Considerations ..................................... 4
8. References ...................................................... 4
   8.1. Normative References ..................................... 4
   8.2. Informative References ................................... 4
9. Authors’ Addresses .......................................... 5

1. Terminology and Conventions

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

This document assumes readers are familiar with the terms and
concepts that are used in [RFC6690], [RFC7252] and
[I-D.ietf-core-resource-directory].

2. Background

The concept of the Resource Directory (RD) is described in
[I-D.ietf-core-resource-directory]. It is defined as a node which
hosts descriptions of resources held on other servers, allowing
lookups to be performed for those resources. The
[I-D.ietf-core-resource-directory] specifies the web interfaces that
a Resource Directory supports in order for devices to discover the RD
and to register, maintain, lookup and remove resources descriptions.

The relevant specification of interfaces in
[I-D.ietf-core-resource-directory] is given using the CoAP protocol
[RFC7252]. For example, all the response codes (i.e. success and
error) for registering and looking up resources are CoAP based. Also
a multicast discovery interface is defined [RFC7390]. However, in
theory, the RD interfaces can also be implemented using HTTP
[RFC7252].

The Core Link Format [RFC6690] describes the format of the payload of
a CoAP Response that carries a set of CoAP URIs. With relation to
the RD, the CoRE Link Format is be used by a device to carry (encode) the set of URIs it wants to register with an RD. Also, the Core Link Format is used to carry (encode) the set of URIs returned by a RD for a lookup query (including the initial multicast discovery request).

3. Proposal

It is proposed that the RD should also support the following additional features:

1. Explicit HTTP interfaces – As explained previously the current CoRE specifications are written explicitly with CoAP examples. The specifications should be expanded to also explicitly support HTTP (e.g. HTTP request and response codes). There may be some RD interfaces, such as multicast and Group Function, that may not be supported by HTTP and those should also be explicitly identified and excluded.

2. Mirror Server – The CoRE WG has previously discussed the concept of a mirror server in relation to supporting sleepy devices. Specifically, [I-D.vial-core-mirror-server] recommends to create a new class of RDs which store the actual resource representations (as opposed to simply storing the URI) in a special type of RD called the Mirror Server. Communicating devices can both lookup the resource, and then also fetch directly the resource representation, from the Mirror Server regardless of the state of the sleepy server.

3. Re-direction to another RD – A given RD may not have the URIs being queried for registered in its database. The given RD should have the capability to re-direct the querying client to another RD which may have the information of interest.

4. URI Ranking – Current Internet search engines (e.g. Google) have extensive methods for ranking the URIs returned to a human initiated search query. For example, the concept of Search Engine Optimization (SEO) has spawned a large industry in the web world for specifically this purpose. The concept of URI ranking (to indicate the "value" of the URI) should also be supported by the RD.

5. Indication of transport protocol – Several proposals exist (e.g. [I-D.silverajan-core-coap-alternative-transports]) in the CoRE WG to support alternative transports (e.g. TCP, SMS) for CoAP beyond the current UDP transport. It would be very useful if search results from a RD indicated the type of transport supported by a given URI.
4. Summary

The proposed set of feature extensions for the RD will improve the constrained environment search capability and make deployments more efficient. These RD feature extensions should be individually considered during the CoRE re-charter discussions. Evolution and forward thinking is required for the CoRE RD, as constantly occurs in the current Internet for HTTP web search engines (e.g. Google).

5. Acknowledgements

TBD.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

Not applicable.

8. References

8.1. Normative References

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


8.2. Informative References

[I-D.silverajan-core-coap-alternative-transports]

[I-D.vial-core-mirror-server]


Authors’ Addresses

Akbar Rahman
InterDigital Communications, LLC

Email: akbar.rahman@interdigital.com

Chonggang Wang
InterDigital Communications, LLC

Email: chonggang.wang@interdigital.com
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 October 2, 2015.

Copyright Notice

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

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

The Constrained Application Protocol (CoAP) [RFC7252] is a web protocol designed for communications between resource constrained nodes. By default, CoAP operates on top of UDP or DTLS, but there is interest in using CoAP also over other types of transports, such as SMS [I-D.becker-core-coap-sms-gprs].

An interesting transport for CoAP could be the WebSocket Protocol [RFC6455]. The WebSocket protocol provides two-way communication between a client and a server after upgrading an HTTP [RFC7230] connection, and may be available in an environment that does not allow transportation of CoAP over UDP. This environment can be, for example, a corporate network with Internet access only via an HTTP proxy, or a CoAP application running inside a web browser without access to connectivity means other than HTTP and WebSockets.

This document specifies how to access resources using CoAP requests and responses over the WebSocket Protocol. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either by communicating CoAP directly with a CoAP server that is accessible over a WebSocket Connection, or via an intermediary that proxies CoAP requests and responses between different transports, such as between WebSockets and UDP.
1.1. Overview

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client seeking to retrieve or update a CoAP resource located at a CoAP server that exposes a WebSocket endpoint (Figure 2). The CoAP client takes the role of the WebSocket client, establishes a WebSocket Connection and sends a CoAP request, to which the CoAP server returns a CoAP response. The WebSocket Connection can be used for any number of requests.

Figure 2: CoAP client (WebSocket client) accesses CoAP server (WebSocket server)
The challenge in this configuration is to identify resource in the namespace of the CoAP server: When the WebSocket Protocol is used by a dedicated client directly (i.e., not from a web page through a web browser), the client can connect to any WebSocket endpoint. This means it is necessary that the client is able to determine both the WebSocket endpoint (identified by a "ws" or "wss" URI) and the path and query of the CoAP resource within that endpoint from the same URI. When the WebSocket Protocol is used from a web page, the choices are more limited [RFC6454], but the challenge persists.

Section 3 proposes a new "coap+ws" URI scheme that identifies both a WebSocket endpoint and a resource within that endpoint as follows:

```
coap+ws://example.org/sensors/temperature?u=Cel
```

```
| ______ | ______ | ______ |
| ______ | ______ | ______ |
| Uri-Path: "sensors" | 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 (Client) ---> (Proxy) ---> (Server)
WebSocket (Client) WebSocke (Server) UDP (Client) UDP (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.

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

Further configurations are possible, including those where a WebSocket Connection is established through an HTTP proxy.

1.2. Terminology

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455] and [RFC7252].

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

2. CoAP over WebSockets

CoAP over WebSockets is intentionally very similar to CoAP as defined over UDP. Therefore, instead of presenting CoAP over WebSockets as a new protocol, this document specifies it as a series of deltas from [RFC7252].

2.1. Opening Handshake

Before CoAP requests and responses can be exchanged, a WebSocket Connection needs to be established as defined in Section 4 of [RFC6455]. The WebSocket client MUST include the subprotocol name "coap.v1" in the list of protocols, which indicates support for the protocol defined in this document. Figure 6 shows an example.
GET /.well-known/coap HTTP/1.1
Host: example.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBzb25jZQ==
Sec-WebSocket-Protocol: coap.v1
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYHvhZBb25jZQ==
Sec-WebSocket-Protocol: coap.v1

Figure 6: Example of an Opening Handshake

2.2. Message Format

Once a WebSocket Connection has been established, CoAP requests and responses can be exchanged as WebSocket messages. Since CoAP uses a binary message format, the messages are transmitted in binary data frames as specified in Sections 5 and 6 of [RFC6455].

The message format is very similar to the format specified for CoAP over UDP [RFC7252]. The differences are as follows:

- Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. This means the "T" and "Message ID" fields in the CoAP message header can be elided.

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

```
0                   1                   2                   3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   R   |  TKL  |      Code     |    Token (TKL bytes) ... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 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 the Token, which is scoped locally to the
connection.

The connection is bi-directional, so requests can be sent both by the
entity that established the connection and the remote host.

Retransmission and deduplication of messages is provided by the
WebSocket Protocol. CoAP over WebSockets therefore does not make a
distinction between Confirmable or Non-Confirmable messages, and does
not provide Acknowledgement or Reset messages.

Since the WebSocket Protocol provides ordered delivery of messages,
the mechanism for reordering detection when observing resources
[I-D.ietf-core-observe] is not needed. The value of the Observe
Option in notifications therefore MAY be empty on transmission and
MUST be ignored on reception.

2.4. Connection Health

When a client does not receive any response for some time after
sending a CoAP request (or, similarly, when a client observes a
resource and it does not receive any notification for some time), the
connection between the WebSocket client and the WebSocket server may
be lost or temporarily disrupted without the client being aware of
it.
To check the health of the WebSocket Connection (and thereby of all active requests, if any), the client can send a Ping frame or an unsolicited Pong frame as specified in Section 5.5 of [RFC6455].

2.5. Closing the Connection

The WebSocket Connection is closed as specified in Section 7 of [RFC6455].

All requests for which the CoAP client has not received a response yet, are cancelled when the connection is closed. If the client observes one or more resource over the WebSocket Connection, then the CoAP server (or intermediary in the role of the CoAP server) MUST remove all entries associated with the client from the lists of observers when the connection is closed.

3. CoAP over WebSockets URIs

For the first configuration discussed in Section 1.1, this document defines two new URI 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].

```plaintext
coap-ws-URI =
   "coap+ws:" "//" host [ ":" port ] path-abempty [ "?" query ]

coap-wss-URI =
   "coap+wss:" "//" host [ ":" port ] path-abempty [ "?" query ]
```

Savolainen, et al. Expires October 2, 2015 [Page 8]
The port component is OPTIONAL; the default for "coap+ws" is port 80, while the default for "coap+wss" is port 443.

Fragment identifiers are not part of the request URI and thus MUST NOT be transmitted in a WebSocket handshake or in the URI options of a CoAP request.

4. Security Considerations

CoAP over WebSockets and CoAP over TLS-secured WebSockets do not introduce additional security issues beyond CoAP and DTLS-secured CoAP respectively [RFC7252].

The security considerations of [RFC6455] apply.

5. IANA Considerations

5.1. URI Scheme Registrations

5.1.1. "coap+ws"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+ws".

URI scheme name.
coap+ws

Status.
Permanent.

URI scheme syntax.
Defined in Section 3.

URI scheme semantics.
The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol.

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Interoperability considerations.
None.

Security considerations.
See Section 4.

Contact.
IETF Chair <chair@ietf.org>

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

References.
This document.

5.1.2. "coap+wss"

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+wss".

URI scheme name.
coap+wss

Status.
Permanent.

URI scheme syntax.
Defined in Section 3.

URI scheme semantics.
The "coap+wss" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket Protocol secured with Transport Layer Security (TLS).

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Interoperability considerations.
None.

Security considerations.
See Section 4.
Contact.
  IETF Chair <chair@ietf.org>

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

References.
  This document.

5.2. WebSocket Subprotocol Registration

This document requests the registration of the subprotocol name "coap.v1" in the WebSocket Subprotocol Name Registry.

Subprotocol Identifier.
  coap.v1

Subprotocol Common Name.
  Constrained Application Protocol (CoAP).

Subprotocol Definition.
  This document.

5.3. Well-Known URI Suffix Registration

This document requests the registration of the Well-Known URI suffix "coap" in the Well-Known URI Registry.

URI suffix.
  coap

Change controller.
  IETF.

Specification document(s).
  This document.

Related information.
  None.

6. Acknowledgements

Thanks to Nadir Javed for helpful comments and discussions that have shaped the document.
7. References

7.1. Normative References


7.2. Informative References


Appendix A.  Examples

This section gives examples for the first two configurations discussed in Section 1.1.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be as follows. Figure 8 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap+ws://example.org/sensors/temperature?u=Cel>, for example, from a resource representation that it retrieved previously.

2. It establishes a WebSocket Connection to the endpoint URI composed of the authority "example.org" and the well-known path "/.well-known/coap", <ws://example.org/.well-known/coap>.

3. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.

4. It waits for the server to return a response.

5. The CoAP client uses the connection for further requests, or the connection is closed.
Figure 8: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI
Figure 9 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource `<coap://[2001:DB8::1]/>`. The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response over the WebSocket connection to the client.

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

Teemu Savolainen
Nokia
Hermiankatu 12 D
Tampere FI-33720
Finland

Email: teemu.savolainen@nokia.com

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

Phone: +49-421-218-63905
Email: hartke@tzi.org

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi
Object Security for CoAP (OSCOAP)
draft-selander-ace-object-security-02

Abstract

This memo presents OSCOAP, a scheme for protection of request and response messages of the Constrained Application Protocol (CoAP), using data object security.

Status of this Memo

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

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

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

Copyright and License Notice

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

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

Table of Contents

1. Introduction ................................................. 4
   1.1 Terminology .............................................. 4
2. Background .................................................. 5
4. Secure Message ................................................ 7
   4.1 Secure Message format ..................................... 8
      4.1.1 Secure Message Header ................................ 8
      4.1.2 Secure Message Body .................................. 9
         4.1.2.1 Secure Signed Message Body ......................... 9
         4.1.2.2 Secure Encrypted Message Body .................... 9
      4.1.3 Secure Message Tag ................................... 9
5. Message Protection ........................................... 10
   5.1 CoAP Message Protection (Mode:COAP) ........................ 10
      5.1.1 The Sig Option ....................................... 10
         5.1.1.1 Option Structure ................................ 10
         5.1.1.2 Integrity Protection and Verification ............. 11
         5.1.1.3 SSM Body ......................................... 11
      5.1.2 The Enc Option ....................................... 12
         5.1.2.1 Option Structure ................................ 12
         5.1.2.2 Encryption and Decryption ......................... 13
         5.1.2.3 SEM Body ......................................... 13
         5.1.2.4 CoAP Message with Enc Option .................... 13
      5.1.3 SM Tag ................................................ 14
   5.2 Payload Only Protection (Mode:PAYL) ........................ 14
   5.3 Replay Protection and Freshness ............................ 15
      5.3.1 Replay Protection ..................................... 15
      5.3.2 Freshness ............................................ 15
6. Security Considerations ....................................... 15
7. Privacy Considerations ........................................ 17
8. IANA Considerations .......................................... 18
9. Acknowledgements ............................................. 18
10. References ................................................... 19
    10.1 Normative References ..................................... 19
    10.2 Informative References ................................... 19
Appendix A. Which CoAP Header Fields and Options to Protect .......... 20
   A.1 CoAP Header Fields ....................................... 20
   A.2 CoAP Options ............................................. 20
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 OSCOAP, a data object based communication security solution complementing DTLS and supporting secure messaging end-to-end across intermediary nodes. OSCOAP may be used in very constrained settings where DTLS cannot be supported. OSCOAP can also be combined with DTLS thus enabling, for example, end-to-end security of CoAP payload in combination with hop-by-hop protection of the entire CoAP message during transport between end-point and intermediary node.

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", and "verify". For "signature", see below.

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

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].
JSON Web Signature (JWS), JOSE Header, JWS Payload, and JWS Signature are defined in [RFC7515]. JSON Web Encryption (JWE), JWE AAD, JWE Ciphertext, and JWE Authentication Tag are defined in [RFC7516].

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.

Note that "signature" as in "JSON Web Signature" and the derived terms "Secure Signed Message" and "Sig" option may refer either to digital signature using private key of an asymmetric key pair, or Message Authentication Code using a shared key. In other occurrences we use the term as defined in [RFC4949], meaning digital signature.

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).

COSE is defined in [I-D.schaad-cose-msg].

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.gerdes-ace-actors]. The focus of this memo is on end-to-end security in constrained environments in the presence of intermediary nodes.

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

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 protecting the CoAP payload (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 correspond to two modes of OSCOAP denoted Mode:PAYL 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 8.7 of [I-D.gerdes-ace-actors]. 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 CoAP payload only protection (Mode:PAYL), the end-to-end security requirements apply to 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.

Note that a Mode:PAYL message may have multiple recipients. 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 between the modes is the actual data that is protected, i.e. CoAP payload data only or also other CoAP message data. This memo specifies a common "Secure Message" format that can be used to wrap either payload only (Mode:PAYL) or also additional selected CoAP message fields (Mode:COAP), 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 [RFC7515], JWE [RFC7516], and COSE [I-D.schaad-cose-msg].

Current CMS and JWx objects are undesirably large for very constrained devices, and 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. First estimates with COSE render more compact objects, see Appendix E for a discussion of message format overhead and minimum message expansion. For example, COSE header for a Message Authentication Code object encodes to 37 bytes, while the same header with JWS results in 74 bytes.

Thus, the candidate message format for use in OSCOAP is COSE [I-D.schaad-cose-msg]. Pending a stable version of COSE this draft uses multiple formats and their terminology to illustrate how the message format is applied and processed. It is the intention to replace these with a profile of one single compact secure message format in a future version of this draft.
None of the message formats listed about provide support for replay protection, but it is noted section 10.10 of [RFC7515] 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.

We use the term Secure Message (SM) format to refer to a content format for cryptographically protected data which includes a unique transaction identifier, and some other common data as specified in Section 4.1.1.

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 to profile JOSE objects to become Secure Message formats. Appendix C shows how to profile COSE objects to become Secure Message formats.

4.1 Secure Message format

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

4.1.1 Secure Message Header

The following parameters SHALL be included in the SM Header:

- **Algorithm.** This parameter identifies the cryptographic algorithm(s) used to protect the Secure Message. In case of SSM it has the same semantics as the JOSE Header Parameter "alg" defined in Section 4.1.1 of [RFC7515]. In case of SEM, "direct key agreement" (corresponding to the JWE "alg" = "dir") is assumed, and the encryption algorithm corresponds to the JOSE Header Parameter "enc" (Section 4.1.2 of [RFC7516]). However, the cipher suites are not limited to AEAD algorithms but also include symmetric key encryption combined with private key signature.

- **Context Identifier.** This parameter identifies the sender and the security context/key(s) used together with the Algorithm to protect the message. For Mode:COAP, the Context Identifier typically identifies the sending party and different resources are identified by their Uri-Path. For Mode:PAYL, the Context Identifier may identify the resource itself. The structure of this identifier is unspecified.
o Sequence Number. The Sequence Number parameter enumerates the Secure Messages protected using the security context identified by the Context Identifier, and is used for replay protection and uniqueness of nonce. The start sequence number SHALL be 0. For a given key, any Sequence Number MUST NOT be used more than once.

The ordered sequence (Sequence Number, Context 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 established 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 [RFC7515]. 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, the encrypted Plaintext as defined by the Algorithm, and Additional Authenticated Data (AAD) which is integrity protected by the Algorithm as defined by the Cipher Suite. 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 / Message Authentication Code 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" and "PAYL". The use of Mode:COAP is signaled with the presence of the options Sig or Enc defined in this section. The differences in SM Body and SM Tag as a function of Mode are described below.

Both formats SSM and SEM defined in the previous section are applicable to both Modes. For any Secure Message Mode, the SEM format SHALL be used by default. Examples of SSM and SEM are given in Appendix F.

5.1 CoAP Message Protection (Mode:COAP)

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 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.

5.1.1 The Sig Option

In order to integrity protect CoAP message exchanges including options and headers, 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 E. 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 [RFC7515] (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 [RFC7515] (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 of SHALL consist of the following data, in this order:

- the 8-bit CoAP header field Code;
- all CoAP options present which are marked as IP in Table 3 (Appendix A), in the order as given by the option number (each
Option with Option Header including delta to previous IP-marked Option which is present); and

- the CoAP Payload (if any).

### 5.1.2 The Enc Option

In order to encrypt and integrity protect CoAP messages, a new CoAP option is introduced: the Enc option, indicating the presence of a SEM Mode: COAP object in the CoAP message, containing the encrypted part of the CoAP message. Endpoints supporting this scheme MUST check for the presence of an Enc option, and verify the SEM as described in 5.1.2.2 before accepting a message as valid.

#### 5.1.2.1 Option Structure

The Enc option indicates that certain CoAP Options and Payload (if present) are encrypted, integrity and replay protected using a Secure Encrypted Message (SEM) with Detached Content (see Section 4.1.2.2). The structure of a CoAP message with an Enc option is described in Section 5.1.2.4.

This option is critical, safe to forward, it is not part of a cache key, and it is not repeatable. Table 2 illustrates the structure of this option.

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>Enc</td>
<td>opaque</td>
<td>0 or 12-TBD</td>
</tr>
</tbody>
</table>

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 E. 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 [RFC7516] (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 [RFC7516] (cf.
Appendix B). In addition, the receiving endpoint SHALL process the
Sequence Number as described in Section 5.3.

NOTE: The explicit steps of the protection and verification procedure
will be included in a future version of this draft.

5.1.2.3 SEM Body

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

- all CoAP Options present which are marked as E in Table 3 (see
Appendix A), in the order as given by the Option number (each
Option with Option Header including delta to previous E-marked
Option); and

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

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

- the 8-bit CoAP header field Code;

- all CoAP options present which are marked as IP and not marked
as E in Table 2 (see Appendix A), in the order as given by the
Option number (each Option with Option Header including delta to
previous such Option).

5.1.2.4 CoAP Message with Enc Option

An unprotected CoAP message is encrypted and integrity protected by
means of an Enc option and a SEM. The structure and format of the protected CoAP message being sent instead of the unprotected CoAP message is now described.

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

- The CoAP header SHALL be the same as the unprotected CoAP message.
- The CoAP options SHALL consist of the unencrypted options of the unprotected CoAP message, and the Enc option. The options shall be formatted as in a CoAP message (each Option with Options Header including delta to previous unencrypted Option).
- If the unprotected CoAP message has no Payload then the Enc option SHALL contain the SEM with Detached Content. If the unprotected CoAP message has Payload, then the SEM option SHALL be empty and the Payload of the CoAP message SHALL be the SEM with Detached Content. The Payload is prefixed by the one-byte Payload Marker (0xFF).

5.1.3 SM Tag

This section describes the SM Tag for Mode:COAP, which applies both to SEM and SSM. The SM Tag is defined in 4.1.3. If the message is a CoAP Request, then EAD SHALL be empty. If the message is a CoAP Response, then EAD SHALL consist of the TID of the associated CoAP Request.

5.2 Payload Only Protection (Mode:PAYL)

Referring to examples 1–3 in Section 2 and requirements a and b in Section 3, the case of only protecting CoAP payload using the Secure Message format is now discussed. This is called Mode:PAYL.

The sending endpoint SHALL wrap the Payload, and the receiving endpoint unwrap the Payload in the relevant SM format (SSM or SEM) Mode:PAYL. 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:PAYL, 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 Mode:PAYL object is indicated by setting the option Content-Format to application/smpayl. A CoAP client may request a response containing such a payload wrapping by setting the option Accept to application/smpayl. (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 supporting OSCOAP SHALL maintain Transaction Identifiers (TIDs) of sent and received Secure Messages (see section 4.1.1).

5.3.1 Replay Protection

An endpoint SHALL maintain a TID and associated security context/key(s) for each other endpoint it receives messages from, and one TID and associated security context/key(s) for protecting sent 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 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 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 these intermediary nodes can read and modify information. The trust model where all participating nodes are considered trustworthy is problematic not only from a privacy perspective but also from a security perspective as the intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture weak.

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

Mode:COAP provides point to point encryption, integrity and replay protection, and freshness of response. Payload as well as relevant options and header field Code are protected.

Mode:PAYL only protects payload and only gives replay protection (not freshness), but allows additional use cases such as point to multi-point interactions including publish-subscribe, reverse proxies and proxy caching of responses. In case of symmetric keys the receiver does not get data origin authentication, which requires a digital signature using a private asymmetric key. Mode:PAYL SHALL NOT be used in cases where the CoAP header field Code needs to be integrity protected.

Blockwise transfers in CoAP [I-D.ietf-core-coap-block] can be applied both to Mode:COAP and Mode:PAYL. With Mode:COAP each block and the Block options are integrity protected. Hence each individual block can be securely verified by the receiver, retransmission securely requested etc. With Mode:PAYL the entire payload is encapsulated in a Secure Message which is partitioned into blocks which are sent with unprotected CoAP. The receiver is able to verify the integrity of the payload but only after the last block containing the signature/MAC is received, and if the verification fails the entire message needs to be resent. However, if the verification succeeds, then the transmission in Mode:PAYL has less computational and packet overhead since only one signature/MAC was generated and sent. As CoAP blockwise transfer with Mode:PAYL is prone to Denial of Service attacks, it should only be used for exchanges where this threat can be mitigated, for example within a local area network where link-layer security is activated.
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 analogous 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 CoAP message protection or payload only 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.gerdes-ace-actors]. The Authorization Server may, for example, authenticate the client on behalf of the server, or provide cryptographic keys or credentials to the client and/or server which can be used in the Secure Message exchange. Similarly, the Authorization Server may, on behalf of the server, notify the client of server supported ciphers, in order to facilitate the usage of OSCOAP in deployments with multiple supported cryptographic algorithms.

The security contexts required for SSM and SEM are different. For a SSM, the security context is essentially Algorithm, Context Identifier, Sequence Number and Key. For a SEM it is also required to have a unique Initialization Vector for each message. The Initialization Vector SHALL be the concatenation of a Salt (4 bytes unsigned integer) and the Sequence Number. The Salt SHOULD be established between sender and receiver before the message is sent, to avoid the overhead of sending it in each message. For example, the Salt may be established by the same means as keys are established. For a SEM, the security context is essentially Algorithm, Context Identifier, Salt, Sequence Number and Key.

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 CoAP options provides additional
protection as to the content and nature of the message exchange.

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

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]]|
```

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

Media Type: application/smpayl

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. We are grateful to Malisa Vucinic for providing helpful and timely comments.
10. References

10.1 Normative References


10.2 Informative References


[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70,
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

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

- CoAP options which are Safe-to-Forward SHALL be integrity protected, the only exception being the security options Enc and Sig. See Table 3.

- Block1, Block2 are Unsafe but not intended to be modified by
intermediaries and hence SHALL be integrity protected.

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

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

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

A.2.1.1 Proxy-Scheme

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

- If there is a Proxy-Scheme present, then the client MUST integrity protect the Proxy-Scheme option.
- If there is no Proxy-Scheme option present the client SHALL integrity protect the Proxy-Scheme option set to the URI scheme used in the message sent.
- The server SHALL insert the Proxy-Scheme option with the name of the URI scheme the message was received with before verifying the integrity.

A.2.1.2 Uri-*

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

The Proxy-Uri option is defined to be an invariant with respect to the following processing (applying to a SSM, for SEM see next section):

- If there is a Proxy-Uri present, then the client MUST integrity protect the Proxy-Uri option and the Uri-* options MUST NOT be integrity protected.
- If there is no Proxy-Uri option present, then the client SHALL compose the full URI from Uri-* options according to the method...
The SM Tag is calculated on the following message, modified compared to what is sent:

- All Uri-* options removed
- A Proxy-Uri option with the full URI included
- The server SHALL compose the URI from the Uri-* options according to the method described in section 6.5 of [RFC7252]. The so obtained URI is placed into a Proxy-Uri option, which is included in the integrity verification.

A.2.2 Encryption

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

- Max-Age, Observe: This information is intended to be read by a proxy.
- Enc, Sig: These are the security-providing options.
- Uri-Host, Uri-Port: This information can be inferred from destination IP address and port.
- Proxy-Uri, Proxy-Scheme: This information is intended to be read by a proxy.

In the case of a SEM, the Proxy-Uri MUST only contain Uri-Host and Uri-Port and MUST NOT contain Uri-Path and Uri-Query because the latter options are not intended to be revealed to a Forward Proxy.

A.2.3 Summary

Table 3 summarizes which options are encrypted and integrity protected, if present.

In a SSM, options marked with "a" and "b" are composed into a URI as described above and included as the Proxy-Uri option which is part of the SSM Body. In a SEM, options marked "a" are composed into a URI as described above and included as the Proxy-Uri option in the SEM Additional Authenticated Data.

```
<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>E</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If-Match</td>
<td>opaque</td>
<td>0-8</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
```
Appendix B. JOSE Profile of SM

This section defines profiles of JWS and JWE complying with the Secure Message format (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 [RFC7515], which corresponds to the parameter Algorithm of the Secure Message.

A JWS is a Secure Message if the JOSE Header includes

- the new parameter "cid" defined in B.4, and
- the new parameter "seq" defined in B.3.

An SSM with Detached Content corresponds to a JWS with 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 [RFC7516], 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 JOSE Header includes

- the new parameter "cid" defined in B.4, and
- the IV contains the Sequence Number and a Salt (see Section 6).

An SEM with Detached Content corresponds to a JWE with 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, corresponding to the Secure Message parameter with the same name (Section 4.1.1), SHALL be an integer represented as a byte string. Only the significant bytes are sent (initial bytes with zeros are removed). The start sequence number SHALL be 0. For a given key, "seq" 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 [RFC7515]), meaning that if a receiver does not understand this parameter it must reject the message.

B.4 "cid" (Context Identifier) Header Parameter

The Context Identifier, corresponding to the Secure Message parameter with the same name (Section 4.1.1), SHALL be a unique byte string identifying the security context of the sending party. The parameter "cid" SHALL be marked as critical.
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, this section defines an optimized bespoke Secure Message format (Section 4.1) called the Compact Secure Message (CSM) format. This is intended as a benchmark for COSE [I-D.schaad-cose-msg].

The Compact Secure Message (CSM) format is depicted in 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      ALG      |   CL    |  SL |             CID               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   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, Context Identifier (CID) and Sequence Number (SEQ). The Header parameters are (compare Table 5):

- **Algorithm (ALG)**. This parameter consists of an encoding of the ciphersuite used in the Secure Message. The encoding is TBD.

- **CID Length (CL)**. This parameter consists of a length indication of the header parameter Context Identifier. The actual length of CID is CL + 1 bytes.

- **SEQ Length (SL)**. This parameter consists of a length indication of the header parameter Sequence Number. The actual length of SEQ is SL + 1 bytes.

- **Context Identifier (CID)**. This parameter identifies the security context/key(s) used to protect the Secure Message. Only the significant bytes are sent (initial bytes with zeros are removed).

- **Sequence Number (SEQ)**. This parameter consists of the sequence number used by the sender of the Secure Message. Only the
significant bytes are sent (initial bytes with zeros are removed).

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameter</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALG</td>
<td>Algorithm</td>
<td>8 bits</td>
</tr>
<tr>
<td>CL</td>
<td>Context Identifier Length</td>
<td>5 bits</td>
</tr>
<tr>
<td>SL</td>
<td>Sequence Number Length</td>
<td>3 bits</td>
</tr>
<tr>
<td>CID</td>
<td>Context Identifier</td>
<td>CL + 1: 1-32 bytes</td>
</tr>
<tr>
<td>SEQ</td>
<td>Sequence Number</td>
<td>SL + 1: 1-8 bytes</td>
</tr>
</tbody>
</table>

Table 5: CSM Header Parameters.
The minimum CSM Header is 4 bytes.

The TID consists of the concatenation of SEQ and CID, 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.

Appendix D. COSE Profile of SM

This section defines a profile of the 00-version of COSE [I-D.schaad-cose-msg] complying with the Secure Message format (see Section 4.1) and supporting the two modes of operation Mode:COAP and Mode:PAYL. In the last subsection we elaborate on possible optimizations.

D.1 COSE_Sign or COSE_mac as Secure Signed Message

SSM corresponds to COSE_MSG msg_type 1 (COSE_Sign) or 3 (COSE_mac).
D.1.1 COSE_Sign as Secure Signed Message

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

The "Headers" field of COSE_Sign MUST contain the field "protected" and this field MUST include the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see section 4.1.1).

The mandatory "signatures" array contains one "COSE_signature" item which contains a "protected" field and the mandatory "signature" field. The "protected" field includes:

- the "alg" parameter which corresponds to the parameter Algorithm of the Secure Message (see section 4.1.1);

- the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see section 4.1.1);

The mandatory "signature" field contains the computed signature value.

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

D.1.2 COSE_mac as Secure Signed Message

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

The "Headers" field of COSE_mac object MUST contain the "protected" field, the "recipient" field and the mandatory "tag" field. The "protected" field MUST include:

- the "alg" parameter which corresponds to the parameter Algorithm of the Secure Message (see section 4.1.1);

- the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see section 4.1.1).

The "recipients" array contains one "COSE_encrypt_a" item (section 5 of [I-D.schaad-cose-msg]), which contains an "unprotected" field that includes:
o the "alg" parameter corresponding to the key encryption algorithm, which in the current version of the draft is assumed to be equal to "dir" (constant). (Appendix A of [I-D.schaad-cose-msg]);

o the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see section 4.1.1);

The mandatory "tag" field contains the MAC value.

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

D.2 COSE_encrypt as Secure Encrypted Message

SEM with AEAD algorithm corresponds to COSE_MSG msg_type 2 (COSE_encrypt). A COSE_MSG of type COSE_encrypt [I-D.schaad-cose-msg] is a Secure Message if its fields are defined as follows (see example in Appendix E.3).

The "Headers" field of COSE_encrypt MUST contain the "protected" field, the "recipient" field, the "cipherText" field and depending on the algorithm used, the "iv" field.

The "iv" corresponds to the Initialization Vector, which contains a Salt (see Section 6) and Sequence Number as defined in section 4.1.1. For some algorithms, it is mandatory to include the "iv" field and hence the Salt is sent in each message.

The "protected" field includes:

o the "alg" parameter which corresponds to the parameter Algorithm of the Secure Message (see section 4.1.1);

o the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see section 4.1.1). This parameter is present only if the "iv" field is not present in the COSE_encrypt structure.

The "recipients" array contains one "COSE_encrypt_a" item as defined in section 5 of [I-D.schaad-cose-msg], which contains an "unprotected" field that includes:

o the "alg" parameter corresponding to the key encryption algorithm, which in the current version of the draft is assumed to be equal to "dir" (constant). (Appendix A of [I-D.schaad-cose-msg]);
The new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see section 4.1.1);

The "cipherText" field contains the encrypted plain text, as defined in section 5 of [I-D.schaad-cose-msg].

A SEM with Detached Content corresponds to a COSE_encrypt with "Headers", "recipients", optionally "iv" and "cipherText" fields; i.e. no "aad" field.

D.3 COSE optimizations

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

1. For COSE_encrypt and COSE_mac, there is a ‘recipient’ field (see section 6 of [I-D.schaad-cose-msg]). This field is intended for a setting when the sender is aware of the recipients of the message, and can wrap keys for these recipients. This is not necessarily true in the use cases targeting constrained devices and thus one possible optimization is to remove the ‘recipient’ field. The Context Identifier "cid" can be carried in the Header, preferably in the protected field, to avoid both protected and unprotected fields causing additional overhead. (An alternative is to define "tid", Transaction Identifier, as an array consisting of "seq" and "cid".)

2. Analogous to other key values, one-byte keys/labels can be assigned to the new parameters defined in this document and cipher suites adapted to constrained device processing. For example: "cid" = 11, "seq" = 12, and "AES-CCM" = 14.

3. The combination of secret key encryption and digital signature is well founded in the use cases. A solution based on wrapping one COSE message into another creates substantial overhead (see difference between modified COSE and CSM in Table 11 of Appendix E.4). A valuable optimization would be to define combined cipher suites and security contexts, and corresponding "alg" and "cid" parameters. An example would be 128-bit AES and particular curve parameters for a 64 bytes ECDSA signature.

4. Digitally signed messages have the largest absolute overhead due to the size of the signature (see Appendices E.2 and E.4).
Whereas certain MACs can be securely truncated, signatures cannot. Signature schemes with message recovery allow some remedy since they allow part of the message to be recovered from the signature itself and thus need not be sent. The effective size of the signature could in this way be considerably reduced, which would have a large impact on the message size (compare size of signature and total overhead in Tables 9 and 11). A valuable optimization is thus to support signature schemes with message recovery.

Appendix E. Comparison of message sizes

This section gives some examples of overhead incurred with JOSE, with the current proposal for COSE at the time of writing [I-D.schaad-cose-msg], and with CSM. CSM should be viewed as a lower bound for COSE. Message sizes are also listed for a modified version of COSE implementing some of the optimizations described in Appendix D.3.

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

The size of the header is shown separately from the size of the MAC/signature, since JWS/JWE has no provisions for truncating it. Compact serialization for both JWS and JWE is assumed. For CSM the encoding of algorithms is assumed as in COSE. An 8-byte Context Identifier and a 3-byte Sequence Number are used throughout all examples. To make it easier to read, COSE objects are represented using CBOR’s diagnostic notation rather than a binary dump.

E.1 SSM: Message Authentication Code

This example is based on HMAC-SHA256, with truncation to 16 bytes. For JWS the following header is used:

{"alg":"HS256","cid":0xa1534e3c5fdc09bd,"seq":0x112233}

which encodes to a size of 74 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 object in COSE gives:
\{(1:3, 2:{1:4, "seq":h'112233'}, 9:{3:{"cid":h'a1534e3c5f09bd', 1:-6}}), 10:MAC\}, where MAC is the truncated 16-byte MAC.

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

In a modified version of COSE, with no 'recipient' field (see section 6 of [I-D.schaad-cose-msg]) and protected "cid" in the header, 1-byte key values are assigned to "cid" and "seq", for example: "cid" = 11 and "seq" = 12. The equivalent COSE object would be:

\{(1:3, 2:{1:4, 12:h'112233', 11:h'a1534e3c5f09bd'}, 10:MAC\}, where MAC is the truncated 16-byte MAC.

This modified COSE object encodes to a total size of 40 bytes.

For CSM the same header is represented by 13 bytes.

Table 6 summarizes these results.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>MAC</th>
<th>Total Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>JWS</td>
<td>74 B</td>
<td>43 B</td>
<td>119 bytes</td>
</tr>
<tr>
<td>COSE</td>
<td>37 B</td>
<td>16 B</td>
<td>53 bytes</td>
</tr>
<tr>
<td>mod-COSE</td>
<td>24 B</td>
<td>16 B</td>
<td>40 bytes</td>
</tr>
<tr>
<td>CSM</td>
<td>13 B</td>
<td>16 B</td>
<td>29 bytes</td>
</tr>
</tbody>
</table>

Table 6: Comparison of JWS, COSE, modified COSE and CSM for HMAC-SHA256.

E.2 SSM: Digital Signature

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

For JWS the following header is used:

{"alg":"ECDSA","cid":0xa1534e3c5f09bd,"seq":0x112233}

which encodes to a size of 74 bytes in Base64url, and the 64 bytes of signature encode to 86 bytes. The concatenation marks add 2 bytes to that in the total overhead.

The same object in COSE gives:

{1:1, 2:{"seq":h’112233’}, 5:{[2:(1:-7,"cid":h’a1534e3c5f09bd’), 6:SIG]}}, where SIG is the 64-byte signature.

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

In a modified version of COSE, 1-byte key values are assigned to "cid" and "seq", for example: "cid" = 11 and "seq" = 12. The equivalent COSE object would be:

{1:1, 2:{12:h’112233’}, 5:{2:(1:-7,11:h’a1534e3c5f09bd’), 6:SIG}, where SIG is the 64-byte signature.

The COSE object encodes to a total size of 94 bytes.

For CSM the same header is represented by 13 bytes.

Table 7 summarizes these results.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>Tag</th>
<th>Total Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>JWS</td>
<td>74 B</td>
<td>86 B</td>
<td>162 bytes</td>
</tr>
<tr>
<td>COSE</td>
<td>36 B</td>
<td>64 B</td>
<td>100 bytes</td>
</tr>
<tr>
<td>mod-COSE</td>
<td>30 B</td>
<td>64 B</td>
<td>94 bytes</td>
</tr>
<tr>
<td>CSM</td>
<td>13 B</td>
<td>64 B</td>
<td>77 bytes</td>
</tr>
</tbody>
</table>

Table 7: Comparison of JWS, COSE, modified COSE and CSM for 64 byte ECDSA signature.

E.3 SEM: Authenticated Encryption

This example is based on both AES-128-CCM-8 and AES-128-GCM. Since the former is not supported by JOSE, we use the latter for comparison between JOSE and COSE.

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 whole IV in the CSM format, but also that the JWE and COSE formats may omit the Sequence Number.

The JWE header
encodes to a size of 72 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.

The corresponding COSE object is:

\[
\{1:2, 2:{1:1}, 7:IV, 4:TAG, 9:\{3:{"cid":h’a1534e3c5fbc09bd’, 1:-6})\}\]

where IV is the 12-byte IV and TAG the 16-byte authentication tag.

The COSE object encodes to a total size of 59 bytes.

Table 8 summarizes these results.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>IV</th>
<th>Tag</th>
<th>Total Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>JWE</td>
<td>72 B</td>
<td>16 B</td>
<td>22 B</td>
<td>113 bytes</td>
</tr>
<tr>
<td>COSE</td>
<td>31 B</td>
<td>12 B</td>
<td>16 B</td>
<td>59 bytes</td>
</tr>
</tbody>
</table>

Table 8: Comparison of JWE and COSE for AES-GCM.

The same calculation have been done using CCM mode instead of GCM, and adding a 3-byte "seq". The COSE object is represented as follows:

\[
\{1:2, 2:{1:"AES-CCM","seq":h’112233’}, 4:TAG, 9:\{3:{"cid":h’a1534e3c5fbc09bd’, 1:-6})\}\]

where TAG is the 16-byte authentication tag.

The COSE object encodes to a total size of 52 bytes.

In a modified version of COSE, the ‘recipient’ field is removed (see section 6 of I-D.schaad-cose-msg) and "cid" is protected in the header. 1-byte key values are assigned to "cid", "seq" and "AES-CCM", for example: "cid" = 11, "seq" = 12 and "AES-CCM" = 14. The equivalent COSE object would be:

\[
\{1:2, 2:{1:14,12:h’112233’}, 4:TAG, 9:\{3:{11:h’a1534e3c5fbc09bd’, 1:-6})\}\]

where TAG is the truncated 8-byte authentication tag.
This modified COSE object encodes to a total size of 39 bytes.

For CSM, the corresponding header for AES-128-CCM-8, including the 8 byte Sequence Number, is represented by 13 bytes and the tag is truncated to 8 Bytes.

Table 9 summarizes these results.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>Tag</th>
<th>Total Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSE</td>
<td>44 B</td>
<td>16 B</td>
<td>60 bytes</td>
</tr>
<tr>
<td>mod-COSE</td>
<td>31 B</td>
<td>8 B</td>
<td>39 bytes</td>
</tr>
<tr>
<td>CSM</td>
<td>13 B</td>
<td>8 B</td>
<td>21 bytes</td>
</tr>
</tbody>
</table>

Table 9: Comparison of COSE, modified COSE and CSM for AES-CCM.

E.4 SEM: Symmetric Encryption + Digital Signature

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

For JWS the following header is used:

```
{"alg":"ECDSA","cid":0xa1534e3c5fdc09bd,"seq":0x112233}
```

which encodes to a size of 74 bytes in Base64url, and the 64 bytes of signature encode to 86 bytes. The concatenation marks add 2 bytes to that in the total overhead.

The payload of the JWS object is a JWE object with the following header:

```
{"alg":"dir","cid":0xa1534e3c5f09bd,"enc":"A128GCM"}
```

which encodes to a size of 72 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.

The total size of the JWS object is 275 bytes.
The same object in COSE gives:

\[
\{1:1, 2:\{\text{"seq"} : \text{h'112233'}, 4:\{1:2, 2:\{1:1\}, 7:\text{IV}, 4:\text{TAG}, 9:\{3:\{\text{"cid"} : \text{h'a1534e3c5f}dc09bd', 1:-6\}\}\}, 5:\{2:\{1:-7,\text{"cid"} : \text{h'a1534e3c5}f}dc09bd', 6:\text{SIG}\}\}\},
\]

where SIG is the 64-byte signature, IV is the 12-byte IV and TAG the 16-byte authentication tag.

The COSE object encodes to a total size of 160 bytes.

Table 10 summarizes these results.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>Sig</th>
<th>Payload</th>
<th>Total Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>JWS</td>
<td>74 B</td>
<td>86 B</td>
<td>113 B</td>
<td>275 bytes</td>
</tr>
<tr>
<td>COSE</td>
<td>37 B</td>
<td>64 B</td>
<td>59 B</td>
<td>160 bytes</td>
</tr>
</tbody>
</table>

Table 10: Comparison of JWS and COSE for nested AES-GCM within ECDSA.

The same calculation have been done using CCM mode instead of GCM, and adding a 3-byte "seq" in the protected header.

The COSE object is represented as follows:

\[
\{1:1, 2:\{\text{"seq"} : \text{h'112233'}, 4:\{1:2, 2:\{1:\"AES-CCM\", \"seq" : \text{h'112233'}\}, 4:\text{TAG}, 9:\{3:\{\text{"cid"} : \text{h'a1534e3c5f}dc09bd', 1:-6\}\}\}, 5:\{2:\{1:-7,\text{"cid"} : \text{h'a1534e3c5f}dc09bd', 6:\text{SIG}\}\}\},
\]

where SIG is the 64-byte signature and TAG is the 16-byte authentication tag.

The COSE object encodes to a total size of 153 bytes.

In a modified version of COSE, the 'recipient' field is removed (see section 6 of [I-D.schaad-cose-msg]) and "cid" is protected in the header. 1-byte key values are assigned to "cid", "seq" and "AES-CCM", for example: "cid" = 11, "seq" = 12 and "AES-CCM" = 14. The equivalent COSE object would be:

\[
\{1:1, 2:\{12:\text{h'112233'}, 4:\{1:2, 2:\{1:14,12:\text{h'112233'}\}, 4:\text{TAG}, 9:\{3:\{11:\text{h'a1534e3c}5f}dc09bd', 1:-6\}\}\}, 5:\{2:\{1:-7,11:\text{h'a1534e3c5f}dc09bd', 6:\text{SIG}\}\},
\]

where SIG is the 64-byte signature and TAG is the 8-byte truncated authentication tag.
This modified COSE object encodes to a total size of 134 bytes.

For CSM we assume that an (AES, ECDSA) cipher suite has been defined, and that the "cid" identifies the context used by both the algorithms. Then the corresponding header is represented by 13 bytes, and the signature by 64 bytes.

Table 11 summarizes these results.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>Sig</th>
<th>Payload</th>
<th>Total Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSE</td>
<td>36 B</td>
<td>64 B</td>
<td>52 B</td>
<td>153 bytes</td>
</tr>
<tr>
<td>mod-COSE</td>
<td>30 B</td>
<td>64 B</td>
<td>39 B</td>
<td>134 bytes</td>
</tr>
<tr>
<td>CSM</td>
<td>13 B</td>
<td>64 B</td>
<td>0 B</td>
<td>77 bytes</td>
</tr>
</tbody>
</table>

Table 11: Comparison of nested AES-CCM within ECDSA (COSE, modified COSE) and combined AES-ECDSA (CSM).

Appendix F. Examples

This section gives examples of how to use the new options and message formats defined in this memo.

F.1 CoAP Message Protection

This section illustrates 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.

F.1.1 Integrity Protection of CoAP Message Exchange

Here is an example of a PUT request/response message exchange passing an intermediate node protected with the Sig option. The example illustrates a client closing a lock and getting a confirmation that the lock is closed. Code, Uri-Path and Payload of the request and Code of the response are integrity protected (and other message fields, see Appendix A).
The Context Identifier is an identifier indicating which security context was used to integrity protect the message, and may be used as an identifier for a secret key or a public key. (It may e.g. be the hash of a public key.)

The server and client can verify that the Sequence Number has not been received and used with this key before, 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 in the Figure).

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.

F.1.2 Additional Encryption of CoAP Message

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

Client | Proxy | Server

+-----+ | | Code: 0.01 (GET)
      | GET  | Token: 0x83
      |      | Enc: SEM {"seq":"000015b7",
      |      |     "cid":"34e3c5fdca1509bd",
      |      |     ["glucose" ... ], ...}

+-----+ Code: 0.01 (GET)
      | GET  Token: 0xbe
      |      Enc: SEM {"seq":"000015b7",
      |      "cid":"34e3c5fdca1509bd",
      |      ["glucose" ... ], ...}

<----- Code: 2.05 (Content)
2.05 | | Token: 0xbe
     | 2.05 | Enc:
     |      | Payload: SEM {"seq":"000015b7",
     |      |     "cid":"c09bda155fd34e3c",
     |      |     [... 220], ...

<----- Code: 2.05 (Content)
2.05 | | Token: 0x83
     | 2.05 | Enc:
     |      | Payload: SEM {"seq":"000015b7",
     |      |     "cid":"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 Context Identifier is a hint to the receiver indicating which security context was used to encrypt and integrity protect the message, and may be used as an identifier for the AEAD secret key. One key is used for each direction of the message transfer.

The server and client can verify that the Sequence Number has not been received and used with this key before, and 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 in the Figure).

F.2 Payload Protection

This section gives examples that illustrate Mode:PAYL. This mode assumes that only the intended receiver(s) has the relevant security context related to the resource. In case of a closed group of recipients of the same object, e.g. in Information-Centric Networking or firmware update distribution, it may be necessary to support symmetric key encryption in combination with digital signature.

F.2.1 Proxy Caching

This examples applies e.g. to closed user groups of a single data source. The 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

```
| RH | GET  | Code: 0.01 (GET) | Token: 0x83 | Proxy-Uri: example.com/temp |
```

```
| RH | GET  | Code: 0.01 (GET) | Token: 0xbe | Uri-Host: example.com |
```
F.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.
This example deviates from encryption (SEM) by default (see Section 6) just to illustrate the SSM in Mode:PAYL. If there is no compelling reason why the payload should be in plaintext, then SEM MUST be used.

F.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 [I-D.seitz-ace-core-authz] and Section 8.4–8.6 of [I-D.gerdes-ace-actors].
Here, C is clearly not trusted with modifying the information, but may need to be involved in mediating the authorization information to the RS, for example, because AS and RS does not have direct connectivity. So end-to-end security is required and object security ("access tokens") is the natural candidate.

This example considers the authorization information to be encapsulated in a SEM Mode:PAYL object, generated by AS. How C accesses the SEM 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 POST to /authorize. 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.

![Figure 12: Protected Transfer of Access Token = SEM (Mode:PAYL)](attachment:fig12.png)

Authors' Addresses

CoAP Communication with Alternative Transports

draft-silverajan-core-coap-alternative-transports-08

Abstract

CoAP has been standardised as an application level REST-based protocol. A single CoAP message is typically encapsulated and transmitted using UDP or DTLS as transports. These transports are optimal solutions for CoAP use in IP-based constrained environments and nodes. However compelling motivation exists for allowing CoAP to operate with other transports and protocols. Examples are M2M communication in cellular networks using SMS, more suitable transport protocols for firewall/NAT traversal, end-to-end reliability and security such as TCP and TLS, or employing proxying and tunneling gateway techniques such as the WebSocket protocol. This draft examines the requirements for conveying CoAP messages to end points over such alternative transports. It also provides a new URI format for representing CoAP resources over alternative transports.

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 22, 2015.

Copyright Notice

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

The Constrained Application Protocol (CoAP) [RFC7252] has been standardised by the CoRE WG as a lightweight, HTTP-like protocol providing a request/response model that constrained nodes can use to communicate with other nodes, be those servers, proxies, gateways, less constrained nodes, or other constrained nodes. CoAP has been defined to utilise UDP and DTLS as transports.

As the Internet evolves by integrating new kinds of networks, services and devices, the need for a consistent, lightweight method for resource representation, retrieval and manipulation becomes...
evident. Owing to its simplicity and low overhead, CoAP is a highly suitable protocol for this purpose. However, communicating CoAP endpoints can reside in networks where end-to-end UDP-based communication can be challenging. These include networks separated by NATs and firewalls, cellular networks in which the Short Messaging Service (SMS) can be utilised as between nodes, or simply situations where an endpoint has no possibility to communicate over UDP. Consequently in addition to UDP and DTLS, alternative transport channels for conveying CoAP messages should be considered.

Extending CoAP over alternative transports allows CoAP implementations to have a significantly larger relevance in constrained as well as non-constrained networked environments: it leads to better code optimisation in constrained nodes and broader implementation reuse across new transport channels. As opposed to implementing new resource retrieval mechanisms, an application in an end-node can continue relying on using CoAP’s REST-based resource retrieval and manipulation for this purpose, while changes in endpoint identification and the transport protocol can be addressed by a transport-specific messaging sublayer. This simplifies development and memory requirements. Resource representations are also visible in an end-to-end manner for any CoAP client. In certain conditions, the processing and computational overhead for conveying CoAP Requests and Responses from one underlying transport to another, would be less than that of an application-level gateway performing protocol translation of individual messages between CoAP and another resource retrieval protocol such as HTTP.

This document first provides scenarios where usage of CoAP over alternative transports is either currently underway, or may prove advantageous in the future. A simple transport type classification for CoAP-capable nodes is provided next. Then a new URI format is described through which a CoAP resource representation can be formulated that expresses transport identification in addition to endpoint information and resource paths. Following that, a discussion of the various transport properties which influence how CoAP Request and Response messages are mapped to transport level payloads, is presented.

This document however, does not touch on application QoS requirements, user policies or network adaptation, nor does it advocate replacing the current practice of UDP-based CoAP communication.
2. Usage Cases

Apart from UDP and DTLS, CoAP usage is being specified for the following environments as of this writing:

2.1. Use of SMS

CoAP messages can be sent via SMS between CoAP end-points in a cellular network [I-D.becker-core-coap-sms-gprs]. A CoAP Request message can also be sent via SMS from a CoAP client to a sleeping CoAP Server as a wake-up mechanism and trigger communication via IP. For this reason, the Open Mobile Alliance (OMA) specifies both UDP and SMS as transports for M2M communication in cellular networks. The OMA Lightweight M2M (LWM2M) protocol being drafted uses CoAP, and as transports, specifies both UDP as well as Short Message Service (SMS) bindings [OMALWM2M]. DTLS is being proposed for securing CoAP messages over SMS between Mobile Stations [I-D.fossati-dtls-over-gsm-sms].

2.2. Use of WebSockets

The WebSocket protocol has been proposed as a transport channel between WebSocket enabled CoAP end-points on the Internet [I-D.savolainen-core-coap-websockets]. This is particularly useful to enable CoAP communication within HTML5 apps and web browsers, especially in smart devices, that do not have any means to use low-level socket interfaces. Embedded client side scripts create new WebSocket connections to various WebSocket-enabled servers, through which CoAP messages can be exchanged. This also allows a browser containing an embedded CoAP server to open a connection to a WebSocket enabled CoAP Mirror Server [I-D.vial-core-mirror-server] to register and update its resources.

2.3. Use of P2P Overlays

[I-D.jimenez-p2psip-coap-reload] specifies how CoAP nodes can use a peer-to-peer overlay network called RELOAD, as a resource caching facility for storing wireless sensor data. When a CoAP node registers its resources with a RELOAD Proxy Node (PN), the node computes a hash value from the CoAP URI and stores it as a structure together with the PN’s Node ID as well as the resources. Resource retrieval by CoAP nodes is accomplished by computing the hash key over the Request URI, opening a connection to the overlay and using its message routing system to contact the CoAP server via its PN.
2.4. Use of TCP and TLS

Using TCP [I-D.tschofenig-core-coap-tcp-tls], allows easier communication between CoAP clients and servers separated by firewalls and NATs. This also allows CoAP messages to be transported over push notification services from a notification server to a client app on a smartphone, that may previously have subscribed to receive change notifications of CoAP resource representations, possibly by using CoAP Observe [I-D.ietf-core-observe].

[I-D.tschofenig-core-coap-tcp-tls] also discusses using TLS as a transport to securely convey CoAP messages over TCP.

3. Node Types based on Transport Availability

The term "alternative transport" in this document thus far has been used to refer to any non-UDP and non-DTLS transport that can convey CoAP messages in its payload. A node however, may in fact possess the capability to utilise CoAP over multiple transport channels at its disposal, simultaneously or otherwise, at any point in time to communicate with a CoAP end-point. Such communication can obviously take place over UDP and DTLS as well. Inevitably, if two CoAP endpoints reside in distinctly separate networks with orthogonal transports, a CoAP proxy node is needed between the two networks so that CoAP Requests and Responses can be exchanged properly.

In [RFC7228], Tables 1, 3 and 4 introduced classification schemes for devices, in terms of their resource constraints, energy limitations and communication power. For this document, in addition to these capabilities, it seems useful to additionally identify devices based on their transport capabilities.

<table>
<thead>
<tr>
<th>Name</th>
<th>Transport Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>Single transport</td>
</tr>
<tr>
<td>T1</td>
<td>Multiple transports, with one or more active at any point in time</td>
</tr>
<tr>
<td>T2</td>
<td>Multiple active transports at all times</td>
</tr>
</tbody>
</table>

Table 1: Classes of Available Transports
Type T0 nodes possess the capability of exactly 1 type of transport channel for CoAP, at all times. These include both active and sleepy nodes, which may choose to perform duty cycling for power saving.

Type T1 nodes possess multiple different transports, and can retrieve or expose CoAP resources over any or all of these transports. However, not all transports are constantly active and certain transport channels and interfaces could be kept in a mostly-off state for energy-efficiency, such as when using CoAP over SMS (refer to section 2.1).

Type T2 nodes possess more than 1 transport, and multiple transports are simultaneously active at all times. CoAP proxy nodes which allow CoAP endpoints from disparate transports to communicate with each other, are a good example of this.

4. CoAP Alternative Transport URI

Based on the usage scenarios as well as the transport classes presented in the preceding sections, this section discusses the formulation of a new URI format for representing CoAP resources over alternative transports.

CoAP is logically divided into 2 sublayers, whereby the upper layer is responsible for the protocol functionality of exchanging request and response messages, while the messaging layer is bound to UDP. These 2 sublayers are tightly coupled, both being responsible for properly encoding the header and body of the CoAP message. The CoAP URI is used by both logical sublayers. For a URI that is expressed generically as

\[ \text{URI} = \text{scheme} \text{"":""} \text{authority path-abempty ["?" query] } \]

a simple example CoAP URI, "coap://server.example.com/sensors/temperature" is interpreted as follows:

```
coap :// server.example.com /sensors/temperature
  \_\_\_/   \_\_\_/   \_\_/     \\
 protocol  endpoint     parameterised
    identifier  identifier  resource  identifier
```

Figure 1: The CoAP URI format
The resource path is explicitly expressed, and the endpoint identifier, which contains the host address at the network-level is also directly bound to the scheme name containing the application-level protocol identifier. The choice of a specific transport for a scheme, however, cannot be embedded with a URI, but is defined by convention or standardisation of the protocol using the scheme. As examples, [RFC5092] defines the 'imap' scheme for the IMAP protocol over TCP, while [RFC2818] requires that the 'https' protocol identifier be used to differentiate using HTTP over TLS instead of TCP.

4.1. Design Considerations

Several ways of formulating a URI which express an alternative transport binding to CoAP, can be envisioned. When such a URI is provided from an application to its CoAP implementation, the URI component containing transport-specific information can be checked to allow CoAP to use the appropriate transport for a target endpoint identifier.

The following design considerations influence the formulation of a new URI expressing CoAP resources over alternative transports:

1. The CoAP Transport URI must conform to the generic syntax for a URI described in [RFC3986]. By ensuring conformance to RFC3986, the need for custom URI parsers as well as resolution algorithms can be obviated. In particular, a URI format needs to be described in which each URI component clearly meets the syntax and percent-encoding rules described.

2. A CoAP Transport URI can be supplied as a Proxy-Uri option by a CoAP end-point to a CoAP forward proxy. This allows communication with a CoAP end-point residing in a network using a different transport. Section 6.4 of [RFC7252] provides an algorithm for parsing a received URI to obtain the request’s options. Conformance to [RFC3986] is also necessary in order for the parsing algorithm to be successful.

3. Request messages sent to a CoAP endpoint using a CoAP Transport URI may be responded to with a relative URI reference, for example, of the form ".../..path/to/resource". In such cases, the requesting endpoint needs to resolve the relative reference against the original CoAP Transport URI to then obtain a new target URI to which a request can be sent to, to obtain a resource representation. [RFC3986] provides an algorithm to establish how relative references can be resolved against a base URI to obtain a target URI. Given this algorithm, a URI format needs to be described in which relative reference resolution does
4. The host component of current CoAP URIs can either be an IPv4 address, an IPv6 address or a resolvable hostname. While the usage of DNS can sometimes be useful for distinguishing transport information (see section 4.3.1), accessing DNS over some alternative transport environments may be challenging. Therefore, a URI format needs to be described which is able to represent a resource without heavy reliance on a naming infrastructure, such as DNS.

4.2. URI format

To meet the design considerations previously discussed, the transport information is expressed as part of the URI scheme component. This is performed by minting new schemes for alternative transports using the form "coap+<transport-name>" and/or "coaps+<transport-name>" where the name of the transport is clearly and unambiguously described. Each scheme name formed in this manner is used to differentiate the use of CoAP, or CoAP using DTLS, over an alternative transport respectively. The endpoint identifier, path and query components together with each scheme name would be used to uniquely identify each resource.

Examples of such URIs are:

- `coap+tcp://[2001:db8::1]:5683/sensors/temperature` for using CoAP over TCP
- `coap+tls://[2001:db8::1]:5683/sensors/temperature` for using CoAP over TLS
- `coaps+sctp://[2001:db8::1]:5683/sensors/temperature` for using CoAP over DTLS over SCTP
- `coap+sms://0015105550101/sensors/temperature` for using CoAP over SMS with the endpoint identifier being a telephone subscriber number
- `coaps+sms://0015105550101/sensors/temperature` for using CoAP over DTLS over SMS with the endpoint identifier being a telephone subscriber number
- `coap+ws://www.example.com/sensors/temperature` for using CoAP over WebSockets
A URI of this format to distinguish transport types is simple to understand and not dissimilar to the CoAP URI format. As the usage of each alternative transport results in an entirely new scheme, IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [I-D.ietf-appsawg-uri-scheme-reg], particularly where permanent URI scheme registration is concerned. CoAP resources transported over UDP or DTLS must conform to Section 6 of [RFC7252] and utilise "coap" or "coaps" for the URI scheme, instead of "coap+udp" or "coap+dtls".

It is also entirely possible for each new scheme to specify its own rules for how resource and transport endpoint information can be presented. However, the URIs and resource representations arising from their usage should meet the URI design considerations and guidelines mentioned in Section 4.1. In addition, each new transport being defined should take into consideration the various transport-level properties that can have an impact on how CoAP messages are conveyed as payload. This is elaborated on in the next section.

5. Alternative Transport Analysis and Properties

In this section the various characteristics of alternative transports for successfully supporting various kinds of functionality for CoAP are considered. CoAP factors lossiness, unreliability, small packet sizes and connection statelessness into its protocol logic. General transport differences and their impact on carrying CoAP messages here are discussed.

Property 1: 1:N communication support.

This refers to the ability of the transport protocol to support broadcast and multicast communication. For example, group communication for CoAP is based on multicasting Request messages and receiving Response messages via unicast [RFC7390]. A protocol such as TCP would be ill-suited for group communications using multicast. Anycast support, where a message is sent to a well defined destination address to which several nodes belong, on the other hand, is supported by TCP.

Property 2: Transport-level reliability.

This refers to the ability of the transport protocol to support properties such as guaranteeing reliability against packet loss, ensuring ordered packet delivery and having error control. When CoAP Request and Response messages are delivered over such transports, the...
CoAP implementations elide certain fields in the packet header. As an example, if the usage of a connection-oriented transport renders it unnecessary to specify the various CoAP message types, the Type field can be elided. For some connection-oriented transports, such as WebSockets, the version of CoAP being used can be negotiated during the opening transfer. Consequently, the Version field in CoAP packets can also be elided.

Property 3: Message encoding.

While parts of the CoAP payload are human readable or are transmitted in XML, JSON or SenML format, CoAP is essentially a low overhead binary protocol. Efficient transmission of such packets would therefore be met with a transport offering binary encoding support. Techniques exist in allowing binary payloads to be transferred over text-based transport protocols such as base-64 encoding. When using SMS as a transport, for example, although binary encoding is supported, Appendix A.5 of [I-D.bormann-coap-misc] indicates binary encoding for SMS may not always be viable. A fuller discussion about performing CoAP message encoding for SMS can be found in Appendix A.5 of [I-D.bormann-coap-misc]

Property 4: Network byte order.

CoAP, as well as transports based on the IP stack use a Big Endian byte order for transmitting packets over the air or wire, while transports based on Bluetooth and Zigbee prefer Little Endian byte ordering for packet fields and transmission. Any CoAP implementation that potentially uses multiple transports has to ensure correct byte ordering for the transport used.

Property 5: MTU correlation with CoAP PDU size.

Section 4.6 of [RFC7252] discusses the avoidance of IP fragmentation by ensuring CoAP message fit into a single UDP datagram. End-points on constrained networks using 6LoWPAN may use blockwise transfers to accommodate even smaller packet sizes to avoid fragmentation. The MTU sizes for Bluetooth Low Energy as well as Classic Bluetooth are provided in Section 2.4 of [I-D.ietf-6lo-btle]. Transport MTU correlation with CoAP messages helps ensure minimal to no fragmentation at the transport layer. On the other hand, allowing a CoAP message to be delivered using a delay-tolerant transport service such as the Bundle Protocol [RFC5050] would imply that the CoAP message may be fragmented (or reconstituted) along various nodes in the DTN as various sized bundles and bundle fragments.

Property 6: Framing
When using CoAP over a streaming transport protocol such as TCP, as opposed to datagram based protocols, care must be observed in preserving message boundaries. Commonly applied techniques at the transport level include the use of delimiting characters for this purpose as well as message framing and length prefixing.

Property 7: Transport latency.

A confirmable CoAP request would be retransmitted by a CoAP end-point if a response is not obtained within a certain time. A CoAP end-point registering to a Resource Directory uses a POST message that could include a lifetime value. A sleepy end-point similarly uses a lifetime value to indicate the freshness of the data to a CoAP Mirror Server. Care needs to be exercised to ensure the latency of the transport being used to carry CoAP messages is small enough not to interfere with these values for the proper operation of these functionalities.

Property 8: Connection Management.

A CoAP endpoint using a connection-oriented transport should be responsible for proper connection establishment prior to sending a CoAP Request message. Both communicating endpoints may monitor the connection health during the Data Transfer phase. Finally, once data transfer is complete, at least one end point should perform connection teardown gracefully.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

New security risks are not envisaged to arise from the guidelines given in this document, for describing a new URI format containing transport identification within the URI scheme component. However, when specific alternative transports are selected for implementing support for carrying CoAP messages, risk factors or vulnerabilities can be present. Examples include privacy trade-offs when MAC addresses or phone numbers are supplied as URI authority components, or if specific URI path components employed for security-specific interpretations are accidentally encountered as false positives. While this document does not make it mandatory to introduce a security mode with each transport, it recommends ascribing meaning to the use of "coap+" and "coaps+" prefixes in the scheme component, with the "coaps+" prefix used for DTLS-based CoAP messages over the alternative transport.
8. Acknowledgements

The draft has benefited greatly from reviews, comments and ideas from
Thomas Fossati, Akbar Rahman, Klaus Hartke, Martin Thomson, Mark
Nottingham, Dave Thaler, Graham Klyne, Carsten Bormann and Markus
Becker.

9. References

9.1. Normative References

[I-D.ietf-appsawg-uri-scheme-reg]
   Thaler, D., Hansen, T., Hardie, T., and L. Masinter,
   "Guidelines and Registration Procedures for URI Schemes",
   draft-ietf-appsawg-uri-scheme-reg-06 (work in progress),
   April 2015.

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

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

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

9.2. Informative References

[BTCorev4.1]
   BLUETOOTH Special Interest Group,
   "BLUETOOTH Specification Version 4.1",
   December 2013.

[I-D.becker-core-coap-sms-gprs]
   Becker, M., Li, K., Kuladinithi, K., and T. Poetsch,
   "Transport of CoAP over SMS",
draft-becker-core-coap-sms-gprs-05 (work in progress),
   August 2014.

[I-D.bormann-coap-misc]
   Bormann, C. and K. Hartke,
   "Miscellaneous additions to CoAP",
draft-bormann-coap-misc-27 (work in progress),
   November 2014.


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


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


Silverajan & Savolainen Expires December 22, 2015
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:
An implementation note here is that some generic URI parsers will fail when encountering a URI such as "coap-at://tcp-[2001:db8::1]/sensors/temperature". Consequently, an equivalent, but parseable URI from the ip6.arpa domain needs to be formulated instead. For [2001:db8::1] using TCP, this would result in the following URL:

```
coap-at://tcp-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.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:
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 = "//" authority path-abempty

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
```

```
/       /    
Prefix  CoAP Resource
```

Figure 2: Prefixing a CoAP URI with TCP transport

Such a representation would result in the URI being decomposed into its constituent components, with the CoAP resource residing within the query component as follows:
Scheme: coap-at

Path: tcp://example.com

Query: coap://example.com/sensors/temperature

The same CoAP resource, if requested over a WebSocket transport, would result the following URI:

coop-at:ws://example.com/endpoint?coap://example.com/sensors/temperature

\____________________________\ \____________________________\
\ Transport-specific \ CoAP Resource \\
\ Prefix \\

Figure 3: Prefixed a CoAP URI with WebSocket transport

While the transport prefix changes, the CoAP resource representation remains the same in the query component:

Scheme: coap-at

Path: ws://example.com/endpoint

Query: coap://example.com/sensors/temperature

The URI format described here overcomes URI aliasing [WWWArchv1] when multiple transports are used, by ensuring each CoAP resource representation remains the same, but is prefixed with different transports. However, against a base URI of this format, resolving relative references of the form "//example.net/sensors/temperature" and "/sensor2/temperature" would again result in target URIs which lose transport-specific information.

Implementation note: While square brackets are disallowed within the path component, the '[' and ']' characters needed to enclose a literal IPv6 address can be percent-encoded into their respective equivalents. The ':' character does not need to be percent-encoded. This results in a significantly simpler URI string compared to section 2.2, particularly for compressed IPv6 addresses.

Additionally, the URI format can be used to specify other similar address families and formats, such as Bluetooth addresses [BTCorev4.1].
A.3. Transport as part of a 'service:' URL scheme

The "service:" URL scheme name was introduced in [RFC2609] and forms the basis of service description used primarily by the Service Location Protocol. An abstract service type URI would have the form "service:<abstract-type>::<concrete-type>"

where <abstract-type> refers to a service type name that can be associated with a variety of protocols, while the <concrete-type> then providing the specific details of the protocol used, authority and other URI components.

Adopting the "service:" URL scheme to describe CoAP usage over alternative transports would be rather trivial. To use a previous example, a CoAP service to discover a Resource Directory and its base RD resource using TCP would take the form


The syntax of the "service:" URL scheme differs from the generic URI syntax and therefore such a representation should be treated as an opaque URI as Section 2.1 of [RFC2609] recommends.

Authors' Addresses

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

Teemu Savolainen
Nokia
Hermiankatu 12 D
FI-33720 Tampere
Finland

Email: teemu.savolainen@nokia.com
CoAP Protocol Negotiation
draft-silverajan-core-coap-protocol-negotiation-00

Abstract

CoAP has been standardised as an application level REST-based protocol. This document introduces a way for CoAP clients and servers to interact with resources by agreeing upon alternate locations as well as transport and protocol configurations.

Status of This Memo

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

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

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

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

Copyright Notice

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

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

The Constrained Application Protocol (CoAP) [RFC7252] provides a lightweight request-response messaging mechanism for retrieving and manipulating resources identified by Uniform Resource Identifiers (URIs). However URIs have a twofold purpose in CoAP: In addition to identifying resources, URIs are also used as locators for origin servers, proxies and endpoints delivering resource representations to clients. Should an origin server wish to serve a resource over multiple transports, a single CoAP URI cannot be used to express the identity of the resource independently of alternate underlying transports or protocol configurations. Similarly, if the server wishes to serve representations of the resource from a different endpoint and path, the URI mechanism is incapable of capturing the relationship between these alternate representations or locations.

This draft proposes a new link format attribute as well as a new link relation type that together enable an origin server to serve a resource from other protocol configurations or endpoints. CoAP clients then interact with an origin server’s CoRE resource discovery interface to obtain a set of links describing alternate locations of resources.

This is useful in the following cases:

1. CoAP clients interacting with Type T1 or T2 CoAP origin servers (see Section 3 of [I-D.silverajan-core-coap-alternative-transports]) either before or during an ongoing transaction to communicate using CoAP over a different protocol configuration or alternative transport.

2. Avoiding URI aliases [WWWArchv1], where a single resource is represented with multiple URIs, without describing relations among the alternate representations.
3. Allowing intermediate nodes such as CoAP-based proxies to intelligently cache and respond to CoAP clients with the same resource representation requested over alternative transports or server endpoints.

4. Ability to separate the CoAP resource paths from web-based CoAP endpoint path in a URI.

2. New Link Attribute and Relation types

A CoAP server wishing to allow interactions with resources from multiple locations or transports can do so by specifying the Transport Type "tt" link attribute, which is an opaque string. Multiple transport types can be included in the value of this parameter, each separated by a space. In such cases, transport types appear in a prioritised list, with the most preferred transport type by the CoAP server specified first and the lowest priority transport type last.

At the same time, each transport type supported by the server is also described with an "altloc" link relation type. The "altloc" relation type specifies a URI (containing the URI scheme, authority and optionally path) providing an alternate endpoint location up to but not including the resource path of a representation.

Both "tt" and "altloc" are optional CoAP features. If supported, they occur at the granularity level of an origin server, ie. they cannot be applied selectively on some resources only. Therefore "altloc" is always anchored at the root resource ("/"). Additionally, the "tt" link attribute and "altloc" relation type can be ignored by unsupported CoAP clients.

(TBD: As type T1 nodes may not have all transports active at all times, should a lifetime value be reflected in server responses?)

3. Examples

Example 1 shows a CoAP server returning all transport types and the alternate resource locations to a CoAP client performing a CoAP Request to ./well-known/core

In this case, the server supplies two different locations to interact with resources using CoAP over TCP. At the same time, the path to the WebSocket endpoint is provided in addition to the FQDN of the server, for using CoAP over WebSockets.

Silverajan Expires September 10, 2015 [Page 3]
REQ: GET /.well-known/core

RES: 2.05 Content
</sensors>;ct=40;title="Sensor Index", tt="tcp ws sms",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<coap+tcp://server.example.com/>;rel="altloc",
<coap+tcp://server.example.net/>;rel="altloc",
<coap+ws://server.example.com/ws-endpoint/>;rel="altloc",
<coap+sms://001234567/>;rel="altloc"

Figure 1: Example of Server response

Example 2 shows a CoAP client actively soliciting a CoAP server for all supported transport types and protocol configurations.

REQ: GET /.well-known/core?tt=* 

RES: 2.05 Content
</sensors>;tt="tcp sms ws"
<coap+tcp://server.example.com/>;rel="altloc",
<coap+tcp://server.example.net/>;rel="altloc",
<coap+ws://server.example.com/ws-endpoint/>;rel="altloc",
<coap+sms://001234567/>;rel="altloc"

Figure 2: CoAP client discovering transports supported by a CoAP server.

Example 3 shows a CoAP client explicitly soliciting support for a specific transport type using a query filter parameter.

REQ: GET /.well-known/core?tt=sms

RES: 2.05 Content
</sensors>;tt="tcp sms ws"
<coap+sms://001234567/>;rel="altloc"

Figure 3: CoAP client looking for a specific transport to use with a CoAP server.
4. IANA Considerations

   New link attributes and link relations need to be registered.

5. Security Considerations

   Probably lots. (TBD)

6. Acknowledgements

   Thanks to Klaus Hartke for comments and reviewing this draft, and Teemu Savolainen for initial discussions about protocol negations and lifetime values.

7. References

7.1. Normative References


7.2. Informative References

   [I-D.silverajan-core-coap-alternative-transports]


   [WWWArchv1]

Author’s Address

   Bilhanan Silverajan
   Tampere University of Technology
   Korkeakoulunkatu 10
   FI-33720 Tampere
   Finland

   Email: bilhanan.silverajan@tut.fi
Abstract

The Hypertext Transfer Protocol (HTTP) has been designed with TCP as an underlying transport protocol. The Constrained Application Protocol (CoAP), which has been inspired by HTTP, has on the other hand been defined to make use of UDP. Therefore, reliable delivery and a simple congestion control and flow control mechanism are provided by the message layer of the CoAP protocol.

A number of environments benefit from the use of CoAP directly over a reliable byte stream that already provides these services. This document defines the use of CoAP over TCP as well as CoAP over TLS.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 12, 2015.

Copyright Notice

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

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP can be used freely - UDP [RFC0768], or DTLS [RFC6347] over UDP, is a good choice for transferring small amounts of data in networks that follow the IP architecture. Some CoAP deployments, however, may have to integrate well with existing enterprise infrastructure, where the use of UDP-based protocols may not be well-received or may even be blocked by firewalls. Middleboxes that are unaware of CoAP usage for IoT can make the use of UDP brittle.

Where NATs are still present, CoAP over TCP can also help with their traversal. NATs often calculate expiration timers based on the transport layer protocol being used by application protocols. Many NATs are built around the assumption that a transport layer protocol
such as TCP gives them additional information about the session life cycle and keep TCP-based NAT bindings around for a longer period. UDP on the other hand does not provide such information to a NAT and timeouts tend to be much shorter, as research confirms [HomeGateway].

Some environments may also benefit from the more sophisticated congestion control capabilities provided by many TCP implementations. (Note that there is ongoing work to add more elaborate congestion control to CoAP as well, see [I-D.bormann-core-cocoa].)

Finally, CoAP may be integrated into a Web environment where the front-end uses CoAP from IoT devices to a cloud infrastructure but the CoAP messages are then transported in TCP between the back-end services. A TCP-to-UDP gateway can be used at the cloud boundary to talk to the UDP-based IoT.

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

The present document describes a shim header that conveys length information about each CoAP message included. Modifications to CoAP beyond the replacement of the message layer (e.g., to introduce further optimizations) are intentionally avoided.

2. Terminology

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

3. Constrained Application Protocol

The interaction model of CoAP over TCP is very similar to the one for CoAP over UDP with the key difference that TCP voids the need to provide certain transport layer protocol features, such as reliable delivery, fragmentation and reassembly, as well as congestion control, at the CoAP level. The protocol stack is illustrated in Figure 1 (derived from [RFC7252], Figure 1).
TCP offers features that are not available in UDP and consequently have been provided in the message layer of CoAP. Since TCP offers reliable delivery, there is no need to offer a redundant acknowledgement at the CoAP messaging layer.

Hence, the only message type transported when using CoAP over TCP is the Non-Confirmable message (NON). By nature of TCP, a NON over TCP is still transmitted reliably. Figure 2 (derived from [RFC7252], Figure 3) shows this message exchange graphically. A UDP-to-TCP gateway will therefore discard all empty messages, such as empty ACKs (after operating on them at the message layer), and re-pack the contents of all non-empty CON, NON, or ACK messages (i.e., those ACK messages that have a piggy-backed response) into NON messages.

Similarly, there is no need to detect duplicate delivery of a message. In UDP CoAP, the Message ID is used for relating acknowledgements to Confirmable messages as well as for duplicate detection. Since the Message ID thus is not meaningful over TCP, it is elided (as indicated by the dashes in Figure 2).

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

Figure 1: The CoAP over TLS/TCP Protocol Stack

Figure 2: NON Message Transmission over TCP.
are marked with the bit combination indicating the NON type (no message layer acknowledgement is expected or even possible). A response is sent back as defined in [RFC7252], as illustrated in Figure 3 (derived from [RFC7252], Figure 6).

![Figure 3: NON Request/Response.](image)

### 4. Message Format

The CoAP message format defined in [RFC7252], as shown in Figure 4, relies on the datagram transport (UDP, or DTLS over UDP) for keeping the individual messages separate.

```
<table>
<thead>
<tr>
<th>Ver</th>
<th>T</th>
<th>TKL</th>
<th>Code</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
```

![Figure 4: RFC 7252 defined CoAP Message Format.](image)

In a stream oriented transport protocol such as TCP, some other form of delimiting messages is needed. For this purpose, CoAP over TCP introduces a length field. Figure 5 shows a 1-byte shim header carrying length information prepending the CoAP message header.
The 'Message Length' field is a 16-bit unsigned integer in network byte order.

-- Alternative L2 --

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

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

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

-- Common for L1 and L2 Alternatives --

The "length" field provides the length of the subsequent CoAP message (including the CoAP header but excluding this message length field) in bytes. T is always the code for NON (1).
The initial byte of the frame contains two nibbles, in a similar way to the CoAP option encoding (Section 3.1 of [RFC7252]). The first nibble is used to indicate the length of the options (including any option delimiter), and the payload (if any); it does not include the Code byte or the Token bytes. The first nibble is interpreted as a 4-bit unsigned integer. A value between 0 and 12 directly indicates the length of the options/payload, in bytes. The other three values have a special meaning:

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

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

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

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

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

```
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Len | TKL | Len+ bytes... |      Code     | TKL bytes ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
[1 1 1 1 1 1 1 1] Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

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

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

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

However, RFC 5246 says that "Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType MAY be coalesced into a single TLSPlaintext record, or a single message MAY be fragmented across several records)."

While the Record Layer provides length information about the encapsulated application data and handshaking payloads, TLS implementations typically do not support an API interface that would provide access to the record layer delimiting information. An additional problem with this approach is that this approach would remove the potential optimization of packing several CoAP messages into one record layer message, which is normally a way to amortize the record layer and MAC overhead over all these messages.

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

One other observation is that the message size limitations defined in Section 4.6 of [RFC7252] are no longer strictly necessary. Consenting implementations may want to interchange messages with payload sizes than 1024 bytes, potentially also obviating the need for the Block protocol [I-D.ietf-core-block]. It must be noted that entirely getting rid of the block protocol is not a generally applicable solution, as:

- a UDP-to-TCP gateway may simply not have the context to convert a message with a Block option into the equivalent exchange without any use of a Block option.
- large messages might also cause undesired head-of-line blocking.

The general assumption is therefore that the block protocol will continue to be used over TCP, even if applications occasionally do exchange messages with payload sizes larger than desirable in UDP.

5. Message Transmission

As CoAP exchanges messages asynchronously over the TCP connection, the client can send multiple requests without waiting for responses. For this reason, and due to the nature of TCP, responses are returned during the same TCP connection as the request. In the event that the connection gets terminated, all requests that have not elicited a
response yet are canceled; clients are free to transmit the request again once a connection is reestablished.

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

6. CoAP URI

CoAP [RFC7252] defines the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. RFC 7252 defines these resources for use with CoAP over UDP.

The present specification introduces two new URI schemes, namely "coap+tcp" and "coaps+tcp". The rules from Section 6 of [RFC7252] apply to these two new URI schemes.

[RFC7252], Section 8 (Multicast CoAP), does not apply to the URI schemes defined in the present specification.

Resources made available via one of the "coap+tcp" or "coaps+tcp" schemes have no shared identity with the other scheme or with the "coap" or "coaps" scheme, even if their resource identifiers indicate the same authority (the same host listening to the same port). The schemes constitute distinct namespaces and, in combination with the authority, are considered to be distinct origin servers.

6.1. coap+tcp URI scheme

Coap-tcp-URI = "coap+tcp:" "//" host [ "::" port ] path-abempty [ "?" query ]

The semantics defined in [RFC7252], Section 6.1, applies to this URI scheme, with the following changes:

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

- The port subcomponent indicates the TCP port at which the TLS server for the CoAP server is located. If it is empty or not given, then the default port 443 is assumed (this is different from the default port for "coaps", i.e., CoAP over DTLS over UDP).
- When CoAP is exchanged over TLS port 443 then the "TLS Application Layer Protocol Negotiation Extension" [RFC7301] MUST be used to allow demultiplexing at the server-side unless out-of-band information ensures that the client only interacts with a server that is able to demultiplex CoAP messages over port 443. This would, for example, be true for many Internet of Things deployments where clients are pre-configured to only ever talk with specific servers. [[_1: Shouldn’t we simply always require ALPN? The protocol should not be defined in such a way that it depends on some undefined pre-configuration mechanism. --cabo]]

7. Security Considerations

This document defines how to convey CoAP over TCP and TLS. It does not introduce new vulnerabilities beyond those described already in the CoAP specification. CoAP [RFC7252] makes use of DTLS 1.2 and this specification consequently uses TLS 1.2 [RFC5246]. CoAP MUST NOT be used with older versions of TLS. Guidelines for use of cipher suites and TLS extensions can be found in [I-D.ietf-dice-profile].

8. IANA Considerations

8.1. Service Name and Port Number Registration

IANA is requested to assign the port number 5683 and the service name "coap+tcp", in accordance with [RFC6335].

Service Name.
coap+tcp

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFCthis]

Port Number.
5683

Similarly, IANA is requested to assign the service name "coaps+tcp", in accordance with [RFC6335]. However, no separate port number is used for "coaps" over TCP; instead, the ALPN protocol ID defined in Section 8.3 is used over port 443.

Service Name.
coaps+tcp

Transport Protocol.
tcp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
Constrained Application Protocol (CoAP)

Reference.
[RFC7301], [RFCthis]

Port Number.
443 (see also Section 8.3 of [RFCthis])

8.2. URI Schemes

This document registers two new URI schemes, namely "coap+tcp" and "coaps+tcp", for the use of CoAP over TCP and for CoAP over TLS over TCP, respectively. The "coap+tcp" and "coaps+tcp" URI schemes can thus be compared to the "http" and "https" URI schemes.

The syntax of the "coap" and "coaps" URI schemes is specified in Section 6 of [RFC7252] and the present document re-uses their semantics for "coap+tcp" and "coaps+tcp", respectively, with the exception that TCP, or TLS over TCP is used as a transport protocol.

IANA is requested to add these new URI schemes to the registry established with [RFC4395].
8.3. ALPN Protocol ID

This document requests a value from the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" created by [RFC7301]:

Protocol:
CoAP

Identification Sequence:
0x63 0x6f 0x61 0x70 ("coap")

Reference:
[RFCthis]

9. Acknowledgements

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Michael Koster, Matthias Kovatsch, Szymon Sasin, and Zach Shelby for their feedback.

10. References

10.1. Normative References

[I-D.ietf-dice-profile]


10.2.  Informative References

[HomeGateway]

[I-D.bormann-core-cocoa]

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


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
Abstract

This document describes a network management interface for constrained devices, called CoMI. CoMI is an adaptation of the RESTCONF protocol for use in constrained devices and networks. It is designed to reduce the message sizes, server code size, and application development complexity. The Constrained Application Protocol (CoAP) is used to access management data resources specified in YANG, or SMIv2 converted to YANG. The payload of the CoMI message is encoded in Concise Binary Object Representation (CBOR).

Note

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 6, 2016.
Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
1.1. Design considerations ...................................... 4
1.2. Terminology ................................................ 5
1.2.1. Tree Diagrams .......................................... 6
2. CoMI Architecture ............................................. 6
2.1. RESTCONF/YANG Architecture ................................. 10
2.2. Compression of data-node instance identifier .............. 10
3. CoAP Interface .................................................. 11
4. MG Function Set ............................................... 13
4.1. Data Retrieval .............................................. 14
4.1.1. GET .................................................... 14
4.1.2. Mapping of the 'select' Parameter ....................... 14
4.1.3. Retrieval Examples ..................................... 15
4.2. Data Editing ................................................ 26
4.2.1. Data Ordering .......................................... 27
4.2.2. POST .................................................. 27
4.2.3. PUT .................................................... 27
4.2.4. PATCH ................................................ 27
4.2.5. DELETE ............................................... 28
4.2.6. Editing Multiple Resources ............................. 28
4.3. Notify functions ............................................ 29
4.4. Use of Block ................................................. 31
4.5. Resource Discovery ......................................... 32
4.6. Error Return Codes ......................................... 34
5. Mapping YANG to CoMI payload ................................. 35
5.1. YANG Hash Generation ..................................... 36
5.2. Re-Hash Error Procedure .................................. 36
5.3. ietf-yang-hash YANG Module ............................... 37
5.4. YANG Re-Hash Examples .................................... 40
5.4.1. Multiple Modules ...................................... 42
5.4.2. Same Module .......................................... 43
1. Introduction

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

The draft [I-D.ietf-netconf-restconf] describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG [RFC6020]. RESTCONF allows access to data resources contained in NETCONF [RFC6241] data-stores. RESTCONF messages can be encoded in XML [XML] or JSON [RFC7159]. The GET method is used to retrieve data resources and the POST, PUT, PATCH, and DELETE methods are used to create, replace, merge, and delete data resources.

A large amount of Management Information Base (MIB) [RFC3418] specifications already exists for monitoring purposes. This data can be accessed in RESTCONF if the server converts the SMIv2 modules to YANG, using the mapping rules defined in [RFC6643].

The CoRE Management Interface (CoMI) is intended to work on standardized data-sets in a stateless client-server fashion. The RESTCONF protocol is adapted and optimized for use in constrained environments, using CoAP instead of HTTP. Standardized data sets promote interoperability between small devices and applications from different manufacturers. Stateless communication is encouraged to keep communications simple and the amount of state information small.
in line with the design objectives of 6lowpan [RFC4944] [RFC6775], RPL [RFC6650], and CoAP [RFC7252].

RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not available in CoAP. HTTP uses TCP which is not recommended for CoAP. The transport protocols available to CoAP are much better suited for constrained networks.

CoMI is low resource oriented, uses CoAP, and only supports the methods GET, PUT, PATCH, POST and DELETE. The payload of CoMI is encoded in CBOR [RFC7049] which is automatically generated from JSON [RFC7159]. CBOR has a binary format and hence has more coding efficiency than JSON. To promote small packets, CoMI uses an additional "data-identifier string-to-number conversion" to minimise CBOR payloads and URI length. It is assumed that the managed device is the most constrained entity. The client might be more capable, however this is not necessarily the case.

Currently, small managed devices need to support at least two protocols: CoAP and SNMP [RFC3411]. When the MIB can be accessed with the CoAP protocol, the SNMP protocol can be replaced with the CoAP protocol. Although the SNMP server size is not huge (see Appendix A), the code for the security aspects of SMIv3 [RFC3414] is not negligible. Using CoAP to access secured management objects reduces the code complexity of the stack in the constrained device, and harmonizes applications development.

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

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

1.1. Design considerations

CoMI supports discovery of resources, accompanied by reading, writing and notification of resource values. As such it is close to the device management of the Open Mobile Alliance described in [OMA]. A comparison between CoMI and LWM2M management can be found in Appendix C. CoMI supports MIB modules which have been translated from SMIv2 to YANG, using [RFC6643]. This mapping is read-only so writable SMIv2 objects need to be converted to YANG using an implementation-specific mapping.
CoMI uses a simple URI to access the management object resources. Complexity introduced by instance selection, or multiple object specification is expressed with uri-query attributes. The choice for uri-query attributes makes the URI structure less context dependent.

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

1.2. Terminology

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

Readers of this specification should be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

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

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

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

The following terms are defined in this document:

YANG hash: CoMI object identifier, which is a 30-bit numeric hash of the YANG object identifier string for the object. When a YANG hash value is printed in a request target URI, error-path or other string, then the lowercase hexadecimal representation is used. Leading zeros are used so the value uses 8 hex characters.

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

Notification-node instance: An instance of a schema node of type notification, specified in a YANG module present in the server. The instance is generated in the server at the occurrence of the corresponding event and appended to the default stream.

The following list contains the abbreviations used in this document.
1.2.1. Tree Diagrams

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

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon ":".
- Ellipsis "..." stands for contents of subtrees that are not shown.

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for the reading and modifying of instrumentation variables used for the management of the instrumented node.
Figure 1: Abstract CoMI architecture

Figure 1 is a high level representation of the main elements of the CoAP management architecture. A client sends requests as payload in packets over the network to a managed constrained node.

Objectives are:

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

The client sets the parameter values in the server at bootstrap and intermittently when operational conditions change.

The constrained network requires the payload to be as small as possible, and the constrained server memory requirements should be as small as possible.

For interoperability it is required that in addition to using the Internet Protocol for data transport:

- The names, type, and semantics of the instrumentation variables are standardized.
- The instrumentation variables are described in a standard language.
- The signature of the CoAP request in the server is standardized.
- The format of the packet payload is standardized.
- The notification from server to client is standardized.

The different numbered components of Figure 1 are discussed according to component number.

1. **YANG specification**: contains a set of named and versioned modules. A module specifies a hierarchy of named and typed resources. A resource is uniquely identified by a sequence of its name and the names of the enveloping resources following the hierarchy order. The YANG specification serves as input to the writers of application and instrumentation code and the humans analysing the returned values (arrow from YANG specification to Variable store). The specification can be used to check the correctness of the CoAP request and do the CBOR encoding.

2. **SMIv2 specification**: A named module specifies a set of variables and "conceptual tables". Named variables have simple types. Conceptual tables are composed of typed named columns. The variable name and module name identify the variable uniquely. There is an algorithm to translate SMIv2 specifications to YANG specifications.

3. **CoAP request**: The CoAP request needs a Universal Resource Identifier (URI) and the payload of the packet to send a request. The URI is composed of the schema, server, path and query and
looks like coap://entry.example.com/<path>?<query>. Fragments are not supported. Allowed operations are PUT, PATCH, GET, DELETE, and POST. New variables can be created with POST when they exist in the YANG specification. The Observe option can be used to return variable values regularly or on event occurrence (notification).

(3.1) CoAP <path>: The path identifies the variable in the form "/mg/<hash-value>".

(3.2) CoAP <query>: The query parameter is used to specify additional (optional) aspects like the module name, list instance, and others. The idea is to keep the path simple and put variations on variable specification in the query.

(3.3) CoAP discovery: Discovery of the variables is done with standard CoAP resource discovery using /.well-known/core with ?rt=/core.mg.

(4) Network packet: The payload contains the CBOR encoding of JSON objects. This object corresponds to the converted RESTCONF message payload.

(5) Retrieval, modification: The server needs to parse the CBOR encoded message and identify the corresponding instances in the Variable store. In addition, this component includes the code for CoAP Observe and block options.

(6) Variable store: The store is composed of two parts: Operational state and Configuration data-store (see Section 2.1). CoMI does not differentiate between variable store types. The Variable store contains data-node instances. Values are stored in the appropriate instances, and or values are returned from the instances into the payload of the packet.

(7) Variable instrumentation: This code depends on implementation of drivers and other node specific aspects. The Variable instrumentation code stores the values of the parameters into the appropriate places in the operational code. The variable instrumentation code reads current execution values from the operational code and stores them in the appropriate instances.

(8) Security: The server MUST prevent unauthorized users from reading or writing any data resources. CoMI relies on DTLS [RFC6347] which is specified to secure CoAP communication.
2.1. RESTCONF/YANG Architecture

CoMI adapts the RESTCONF architecture so data exchange and implementation requirements are optimized for constrained devices.

The RESTCONF protocol uses a unified data-store to edit conceptual data structures supported by the server. The details of transaction preparation and non-volatile storage of the data are hidden from the RESTCONF client. CoMI also uses a unified data-store, to allow stateless editing of configuration variables and the notification of operational variables.

The child schema nodes of the unified data-store include all the top-level YANG data nodes in all the YANG modules supported by the server. The YANG data structures represent a hierarchy of data resources. The client discovers the list of YANG modules, and important conformance information such as the module revision dates, YANG features supported, and YANG deviations required. The individual data nodes are discovered indirectly by parsing the YANG modules supported by the server.

The YANG data definition statements contain a lot of information that can help automation tools, developers, and operators use the data model correctly and efficiently. The YANG definitions and server YANG module capability advertisements provide an "API contract" that allow a client to determine the detailed server management capabilities very quickly. CoMI allows access to the same data resources as a RESTCONF server, except the messages are optimized to reduce identifier and payload size.

RESTCONF uses a simple algorithmic mapping from YANG to URI syntax to identify the target resource of a retrieval or edit operation. A client can construct operations or scripts using a predictable syntax, based on the YANG data definitions. The target resource URI can reference a data resource instance, or the data-store itself (to retrieve the entire data-store or create a top-level data resource instance). CoMI uses a compression algorithm to reduce the size of the data-node instance identifier (see Section 2.2.

2.2. Compression of data-node instance identifier

The RESTCONF protocol uses the full path of the desired data resource in the target resource URI. The JSON encoding will include the module name string to specify the YANG module. If a representation of the target resource is included in the request or response message in RESTCONF messages, then the data definition name string is used to identify each node in the message. The module namespace (or name) may also be present in these identifiers.
In order to greatly reduce the size of identifiers used in CoMI, numeric object identifiers are used instead of these strings. The specific encoding of the object identifiers is not hard-wired in the protocol.

YANG Hash is the default encoding for object identifiers. This encoding is considered to be "unstructured" since the particular values for each object are determined by a hash algorithm. It is possible for 2 different objects to generate the same hash value. If this occurs, then the client and server will both need to rehash the colliding object identifiers to new unused hash values.

In order to eliminate the need for rehashing, CoMI allows for alternate "structured" object identifier encoding formats. Structured object identifier MUST be managed such that no object ID collisions are possible, and therefore no rehash procedures are needed. Structured object identifiers can also be selected to minimize the size of a subset of the object identifiers (e.g., the most requested objects).

In Section 4.5 the discovery of the object ID compression scheme is described.

3. CoAP Interface

In CoAP a group of links can constitute a Function Set. The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Function Set. CoMI end-points that implement the CoMI management protocol support at least one discoverable management resource of resource type (rt): core.mg, with path: /mg, where mg is short-hand for management. The name /mg is recommended but not compulsory (see Section 4.5).

The path prefix /mg has resources accessible with the following five paths:

/mg:  YANG-based data with path "/mg" and using CBOR content encoding format. This path represents a data-store resource which contains YANG data resources as its descendant nodes. All identifiers referring to YANG data nodes within this path are encoded as YANG hash values (see Section 5.5).

/mg/mod.uri:  URI identifying the location of the server module information, with path "/mg/mod.uri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG hash values.
/mg/mod.set:  String identifying the module set ID in use by the server, which is defined as the ‘module-set-id’ leaf in the ietf-yang-library module. This resource MUST change to a new value when the set of YANG modules in use by the server changes.

/mg/num.typ:  String identifying the object ID numbering scheme used by the CoMI server. The only value defined in this document is ‘yanghash’ to indicate that the YANG Hash numbering scheme defined in this document is used. It is possible for other object numbering schemes to be defined outside the scope of this document.

/mg/srv.typ:  String identifying the CoMI server type. The value ‘ro’ indicates that the server is a read-only server and no editing operations are supported. A read-only server is not required to provide YANG deviation statements for any writable YANG data nodes. The value ‘rw’ indicates that the server is a read-write server and editing operations are supported. A read-write server is required to provide YANG deviation statements for any writable YANG data nodes that are not fully implemented.

/mg/yh.uri:  URI indicating the location of the server YANG hash information if any objects needed to be re-hashed by the server. It has the path "/mg/yh.uri" and is encoded in CBOR format. The "ietf-yang-hash" module of Section 5.3 is used to define the syntax and semantics of this data structure. This YANG data is encoded with plain identifier strings, not YANG hash values. The server will only have this resource if there are any objects that needed to be re-hashed due to a hash collision.

/mg/stream:  String identifying the default stream resource to which YANG notification instances are appended. Notification support is optional, so this resource will not exist if the server does not support any notifications.

The mapping of YANG data node instances to CoMI resources is as follows: A YANG module describes a set of data trees composed of YANG data nodes. Every root of a data tree in a YANG module loaded in the CoMI server represents a resource of the server. All data root descendants represent sub-resources.

The resource identifiers of the instances of the YANG specifications are YANG hash values, as described in Section 5.1. When multiple instances of a list node exist, the instance selection is described in Section 4.1.3.4
The profile of the management function set, with IF=core.mg, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

<table>
<thead>
<tr>
<th>name</th>
<th>path</th>
<th>rt</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>/mg</td>
<td>core.mg</td>
<td>n/a</td>
</tr>
<tr>
<td>Data</td>
<td>/mg</td>
<td>core.mg.data</td>
<td>application/cbor</td>
</tr>
<tr>
<td>Module Set URI</td>
<td>/mg/mod.uri</td>
<td>core.mg.moduri</td>
<td>application/cbor</td>
</tr>
<tr>
<td>Module Set ID</td>
<td>/mg/mod.set</td>
<td>core.mg.modset</td>
<td>application/cbor</td>
</tr>
<tr>
<td>Numbering</td>
<td>/mg/num.typ</td>
<td>core.mg.num-type</td>
<td>application/cbor</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server Type</td>
<td>/mg/srv.typ</td>
<td>core.mg.srv-type</td>
<td>application/cbor</td>
</tr>
<tr>
<td>YANG Hash Info</td>
<td>/mg/yh.uri</td>
<td>core.mg.yang-hash</td>
<td>application/cbor</td>
</tr>
<tr>
<td>Events</td>
<td>/mg/stream</td>
<td>core.mg.stream</td>
<td>application/cbor</td>
</tr>
</tbody>
</table>

4. MG Function Set

The MG Function Set provides a CoAP interface to perform a subset of the functions provided by RESTCONF.

A subset of the operations defined in RESTCONF are used in CoMI:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieve the data-store resource or a data resource</td>
</tr>
<tr>
<td>POST</td>
<td>Create a data resource</td>
</tr>
<tr>
<td>PUT</td>
<td>Create or replace a data resource</td>
</tr>
<tr>
<td>PATCH</td>
<td>Replace a data resource partially</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete a data resource</td>
</tr>
</tbody>
</table>
4.1. Data Retrieval

4.1.1. GET

One or more instances of data resources are retrieved by the client with the GET method. The RESTCONF GET operation is supported in CoMI. The same constraints apply as defined in section 3.3 of [I-D.ietf-netconf-restconf]. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [I-D.ietf-core-block] is used, as explained in more detail in Section 4.4.

There are two query parameters for the GET method. A CoMI server MUST implement the keys parameter and MAY implement the select parameter to allow common data retrieval filtering functionality.

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keys</td>
<td>Request to select instances of a YANG definition</td>
</tr>
<tr>
<td>select</td>
<td>Request selected sub-trees from the target</td>
</tr>
<tr>
<td></td>
<td>resource</td>
</tr>
</tbody>
</table>

The "keys" parameter is used to specify a specific instance of the resource. When keys is not specified, all instances are returned. When no or one instance of the resource exists, the keys parameter is not needed.

4.1.2. Mapping of the ‘select’ Parameter

RESTCONF uses the ‘select’ parameter to specify an expression which can represent a subset of all data nodes within the target resource [I-D.ietf-netconf-restconf]. This parameter is useful for filtering sub-trees and retrieving only a subset that a managing application is interested in.

However, filtering is a resource intensive task and not all constrained devices can be expected to have enough computing resources such that they will be able to successfully filter and return a subset of a sub-tree. This is especially likely to be true with Class 0 devices that have significantly lesser RAM than 10 KiB
[RFC7228]. Since CoMI is targeted at constrained devices and networks, only a limited subset of the ’select’ parameter is used here.

Unlike the RESTCONF ’select’ parameter, CoMI does not use object names in "XPath" or "path-expr" format to identify the subset that needs to be filtered. Parsing XML is resource intensive for constrained devices [management] and using object names can lead to large message sizes. Instead, CoMI utilizes the YANG hashes described in Section 5 to identify the sub-trees that should be filtered from a target resource. Using these hashes ensures that a constrained node can identify the target sub-tree without expending many resources and that the messages generated are also efficiently encoded.

The implementation of the ’select’ parameter is already optional for constrained devices, however, even when implemented it is expected to be a best effort feature, rather than a service that nodes must provide. This implies that if a node receives the ’select’ parameter specifying a set of sub-trees that should be returned, it will only return those that it is able to.

4.1.3. Retrieval Examples

In all examples the path is expressed in readable names and as a hash value of the name (where the hash value in the payload is expressed as a hexadecimal number, and the hash value in the URL as a base64 number). The examples in this section use a JSON payload with one or more entries describing the pair (identifier, value). CoMI transports the CBOR format to transport the equivalent contents. The CBOR syntax of the payloads is specified in Section 5.

4.1.3.1. Single instance retrieval

A request to read the values of instances of a management object or the leaf of an object is sent with a confirmable CoAP GET message. A single object is specified in the URI path prefixed with /mg.

Using for example the clock container from [RFC7317], a request is sent to retrieve the value of clock/current-datetime specified in module system-state. The answer to the request returns a (identifier, value) pair.
REQ: GET example.com/mg/system-state/clock/current-datetime

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

The YANG hash value for 'current-datetime' is calculated by constructing the schema node identifier for the object:
/sys:system-state/sys:clock/sys:current-datetime

The 30 bit murmur3 hash value is calculated on this string (0x15370408 and VNwQI). The request using this hash value is shown below:

REQ: GET example.com/mg/VNwQI

RES: 2.05 Content (Content-Format: application/cbor)
{
    0x15370408 : "2014-10-26T12:16:31Z"
}

The specified object can be an entire object. Accordingly, the returned payload is composed of all the leaves associated with the object. Each leaf is returned as a (YANG hash, value) pair. For example, the GET of the clock object, sent by the client, results in the following returned payload sent by the managed entity:

REQ: GET example.com/mg/system-state/clock
(Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
    "clock/boot-datetime" : "2014-10-21T03:00:00Z"
}

The YANG hash values for 'clock', 'current-datetime', and 'boot-datetime' are calculated by constructing the schema node identifier for the objects, and then calculating the 30 bit murmur3 hash values (shown in parenthesis):
/sys:system-state/sys:clock (0x2eb2fa3b and usvo7)
/sys:system-state/sys:clock/sys:current-datetime (0x15370408)
/sys:system-state/sys:clock/sys:boot-datetime (0x1fa25361)
The request using the hash values is shown below:

REQ: GET example.com/mg/usvo7
(Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
  0x15370408 : "2014-10-26T12:16:51Z",
  0x1fa25361 : "2014-10-21T03:00:00Z"
}

4.1.3.2. Multiple instance retrieval

A "list" node can have multiple instances. Accordingly, the returned payload is composed of all the instances associated with the list node. Each instance is returned as a (identifier, value) pair. The "keys" query parameter is used to identify a specific list instance by specifying a given index value (see Section 4.1.3.4).

For example, the GET of the /interfaces/interface/ipv6/neighbor instance identified with interface index "eth0" [RFC7223], sent by the client, results in the following returned payload sent by the managed entity:

REQ: GET example.com/mg/interfaces/interface/ipv6/neighbor?keys=eth0
(Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
  "neighbor": [
    {
      "ip" : "fe80::200:ff:fe21:67cf",
      "link-layer-address" : "00:00:10:01:23:45"
    },
    {
      "ip" : "fe80::200:ff:fe21:6708",
      "link-layer-address" : "00:00:10:54:32:10"
    },
    {
      "ip" : "fe80::200:ff:fe21:88ee",
      "link-layer-address" : "00:00:10:98:76:54"
    }
  ]
}
The YANG hash values for `neighbor`, `ip`, and `link-layer-address` are calculated by constructing the schema node identifier for the objects, and then calculating the 30 bit murmur3 hash values (shown in parenthesis):

```
/if:interfaces/if:interface/ip:ipv6/ip:neighbor (0x2354bc49 and jVLxJ)
(0x20b8907e and guJB_)
(0x16f47fd8)
```

The request using the hash values is shown below:

```text
REQ: GET example.com/mg/jVLxJ?keys=eth0
(Content-Format: application/cbor)
```

```json
RES: 2.05 Content (Content-Format: application/cbor)
{
  0x2354bc49 : [
    
    0x20b8907e : "fe80::200:f8ff:fe21:67cf",
    0x16f47fd8 : "00:00::10:01:23:45"
  ],
  
    0x20b8907e : "fe80::200:f8ff:fe21:6708",
    0x16f47fd8 : "00:00::10:54:32:10"
  ],
  
    0x20b8907e : "fe80::200:f8ff:fe21:88ee",
    0x16f47fd8 : "00:00::10:98:76:54"
  ]
}
```

4.1.3.3. Access to MIB Data

The YANG translation of the SMI specifying the `ipNetToMediaTable` [RFC4293] yields:
The following example shows an "ipNetToPhysicalTable" with 2 instances, using JSON encoding:
{  
    "IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry" : [  
    
    
    },  
    
    
    ]  
}  

The YANG hash values for ‘ipNetToPhysicalEntry’ and its child nodes are calculated by constructing the schema node identifier for the objects, and then calculating the 30 bit murmur3 hash values (shown in parenthesis):
The following example shows a request for the entire `ipNetToPhysicalTable`. Since all the instances are requested, no "keys" query parameter is needed.
REQ: GET example.com/mg/wt7w_

RES: 2.05 Content (Content-Format: application/cbor)
{
  0x1067f289 : [
    {
      0x00d38564 : 1,
      0x2745e222 : "ipv4",
      0x387804eb : "10.0.0.51",
      0x1a51514a : "00:00:10:01:23:45",
      0x03f95578 : "2333943",
      0x24ade115 : "static",
      0x09e640ef : "reachable",
      0x3b5c1ab6 : "active"
    },
    {
      0x00d38564 : 1,
      0x2745e222 : "ipv4",
      0x387804eb : "9.2.3.4",
      0x1a51514a : "00:00:10:54:32:10",
      0x03f95578 : "2329836",
      0x24ade115 : "dynamic",
      0x09e640ef : "unknown",
      0x3b5c1ab6 : "active"
    }
  ]
}

4.1.3.4. The 'keys' Query Parameter

There is a mandatory query parameter that MUST be supported by servers called "keys". This parameter is used to specify the key values for an instance of an object identified by a YANG hash value. Any key leaf values of the instance are passed in order. The first key leaf in the top-most list is the first key encoded in the 'keys' parameter.

The key leafs from top to bottom and left to right are encoded as a comma-delimited list. If a key leaf value is missing then all values for that key leaf are returned.

Example: In this example exactly 1 instance is requested from the ipNetToPhysicalEntry (from a previous example).
An example illustrates the syntax of keys query parameter. In this example the following YANG module is used:

```
module foo-mod {
    namespace foo-mod-ns;
    prefix foo;

    list A {
        key "key1 key2";
        leaf key1 { type string; }
        leaf key2 { type int32; }
    } list B {
        key "key3";
        leaf key3 { type string; }
        leaf col1 { type uint32; }
    }
}
```

The path identifier for the leaf "col1" is the following string:

```
/foo:A/foo:B/foo:col1
```

The YANG hash for this identifier string has values: 0xa9abdcca and pq9zK).
The following string represents the RESTCONF target resource URI expression for the "coll" leaf for the key values "top", 17, and "group1":

/restconf/data/foo-mod:A="top",17/B="group1"/coll

The following string represents the CoMI target resource identifier for the same instance of the "coll" leaf:

/mg/pq9zK?keys="top",17,"group1"

4.1.3.5. The 'select' Query Parameter

The select parameter is used along with the GET method to provide a sub-tree filter mechanism. A list of YANG hashes that should be filtered is provided along with a list of keys identifying the instances that should be returned. When the keys parameter is used together with the select, the key values are added in brackets without using the "keys=" text.

The following example shows an "ipNetToPhysicalTable" (from a previous example) with 4 instances, using JSON encoding:
```json
{"IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry": [
  {
    "ipNetToPhysicalIfIndex": 1,
    "ipNetToPhysicalNetAddressType": "ipv4",
    "ipNetToPhysicalNetAddress": "10.0.0.51",
    "ipNetToPhysicalPhysAddress": "00:00:10:01:23:45",
    "ipNetToPhysicalLastUpdated": "2333943",
    "ipNetToPhysicalType": "static",
    "ipNetToPhysicalState": "reachable",
    "ipNetToPhysicalRowStatus": "active"
  },
  {
    "ipNetToPhysicalIfIndex": 1,
    "ipNetToPhysicalNetAddressType": "ipv4",
    "ipNetToPhysicalNetAddress": "9.2.3.4",
    "ipNetToPhysicalPhysAddress": "00:00:10:54:32:10",
    "ipNetToPhysicalLastUpdated": "2329836",
    "ipNetToPhysicalType": "dynamic",
    "ipNetToPhysicalState": "unknown",
    "ipNetToPhysicalRowStatus": "active"
  },
  {
    "ipNetToPhysicalIfIndex": 2,
    "ipNetToPhysicalNetAddressType": "ipv4",
    "ipNetToPhysicalNetAddress": "10.24.2.53",
    "ipNetToPhysicalPhysAddress": "00:00:10:28:19:CA",
    "ipNetToPhysicalLastUpdated": "2124368",
    "ipNetToPhysicalType": "static",
    "ipNetToPhysicalState": "unknown",
    "ipNetToPhysicalRowStatus": "active"
  },
  {
    "ipNetToPhysicalIfIndex": 3,
    "ipNetToPhysicalNetAddressType": "ipv4",
    "ipNetToPhysicalNetAddress": "192.168.2.12",
    "ipNetToPhysicalPhysAddress": "00:00:10:29:11:32",
    "ipNetToPhysicalLastUpdated": "1925384",
    "ipNetToPhysicalType": "dynamic",
    "ipNetToPhysicalState": "reachable",
    "ipNetToPhysicalRowStatus": "active"
  }
]}
```
Data may be retrieved using the select query parameter in the following way:

REQ: GET example.com/mg/?select=wt7w_(ipv4,reachable)

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

{ 0x1067f289 : [ 0x00d38564 : 1, 0x2745e222 : "ipv4", 0x387804eb : "10.0.0.51", 0x1a51514a : "00:00:10:01:23:45", 0x03f95578 : "2333943", 0x24ade115 : "static", 0x09e640ef : "reachable", 0x3b5c1ab6 : "active" ],
[ 0x00d38564 : 3, 0x2745e222 : "ipv4", 0x387804eb : "192.168.2.12", 0x1a51514a : "00:00:10:29:11:32", 0x03f95578 : "1925384", 0x24ade115 : "dynamic", 0x09e640ef : "reachable", 0x3b5c1ab6 : "active" ]
}

In this example exactly 2 instances are returned as response from the ipNetToPhysicalTable because both those instances match the provided keys.

Supposing there were multiple YANG hashes with their own sets of keys that were to be filtered, the select query parameter can be used to retrieve results from these in one go as well. The following string represents the CoMI target resource identifier when multiple YANG hashes, with their own sets of keys are queried:

/mg/?select=hash1(hash1-key1,hash1-key2,...),hash2(hash2-key1)...

4.2. Data Editing

CoMI allows data-store contents to be created, modified and deleted using CoAP methods.
Data-editing is an optional feature. The server will indicate its editing capability with the "/core.rg.srv-type resource type. If the value is 'rw' then the server supports editing operations. If the value is 'ro' then the server does not support editing operations.

4.2.1. Data Ordering

A CoMI server is not required to support entry insertion of lists and leaf-lists that are ordered by the user (i.e., YANG statement "ordered-by user"). The ‘insert’ and ‘point’ query parameters from RESTCONF are not used in CoMI.

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as arrays so messages will preserve their order.

4.2.2. POST

Data resource instances are created with the POST method. The RESTCONF POST operation is supported in CoMI, however it is only allowed for creation of data resources. The same constraints apply as defined in section 3.4.1 of [I-D.ietf-netconf-restconf]. The operation is mapped to the POST method defined in section 5.8.2 of [RFC7252].

There are no query parameters for the POST method.

4.2.3. PUT

Data resource instances are created or replaced with the PUT method. The PUT operation is supported in CoMI. A request to set the values of instances of an object/leaf is sent with a confirmable CoAP PUT message. The Response is piggybacked to the CoAP ACK message corresponding with the Request. The same constraints apply as defined in section 3.5 of [I-D.ietf-netconf-restconf]. The operation is mapped to the PUT method defined in section 5.8.3 of [RFC7252].

There are no query parameters for the PUT method.

4.2.4. PATCH

Data resource instances are partially replaced with the PATCH method [I-D.vanderstok-core-patch]. The PATCH operation is supported in CoMI. A request to set the values of instances of a subset of the values of the resource is sent with a confirmable CoAP PATCH message. The Response is piggybacked to the CoAP ACK message corresponding with the Request. The same constraints apply as defined in section
3.5 of [I-D.ietf-netconf-restconf]. The operation is mapped to the PATCH method defined in [I-D.vanderstok-core-patch].

There are no query parameters for the PATCH method.

4.2.5. DELETE

Data resource instances are deleted with the DELETE method. The RESTCONF DELETE operation is supported in CoMI. The same constraints apply as defined in section 3.7 of [I-D.ietf-netconf-restconf]. The operation is mapped to the DELETE method defined in section 5.8.4 of [RFC7252].

There are no optional query parameters for the PUT method.

4.2.6. Editing Multiple Resources

Editing multiple data resources at once can allow a client to use fewer messages to make a configuration change. It also allows multiple edits to all be applied or none applied, which is not possible if the data resources are edited one at a time.

It is easy to add multiple entries at once. The "PATCH" method can be used to simply patch the parent node(s) of the data resources to be added. If multiple top-level data resources need to be added, then the data-store itself ('/mg') can be patched.

If other operations need to be performed, or multiple operations need to be performed at once, then the YANG Patch [I-D.ietf-netconf-yang-patch] media type can be used with the PATCH method. A YANG patch is an ordered list of edits on the target resource, which can be a specific data node instance, or the data-store itself. The resource type used by YANG Patch is 'application/yang.patch'. A status message is returned in the response, using resource type 'application/yang.patch.status'.

The following YANG tree diagram describes the YANG Patch structure. Each 'edit' list entry has its own operation, sub-resource target, and new value (if needed).
The YANG Hash values for the YANG Patch request objects are calculated as follows:

- 0b346308: /ypatch:yang-patch
- 29988080: /ypatch:yang-patch/ypatch:patch-id
- 0c258737: /ypatch:yang-patch/ypatch:comment
- 316beed6: /ypatch:yang-patch/ypatch:edit
- 2f51f9f7: /ypatch:yang-patch/ypatch:edit/ypatch:edit-id
- 28f4669e: /ypatch:yang-patch/ypatch:edit/ypatch:operation
- 387d0cd8: /ypatch:yang-patch/ypatch:edit/ypatch:point
- 1d86d302: /ypatch:yang-patch/ypatch:edit/ypatch:value

Refer to [I-D.ietf-netconf-yang-patch] for more details on the YANG Patch request and response contents.

4.3. Notify functions

Notification by the server to a selection of clients when an event occurs in the server is an essential function for the management of servers. CoMI allows events specified in YANG [RFC5277] to be notified to a selection of requesting clients. There is one, so-called "default", stream in a CoMI server. The /mg/stream resource identifies the default stream. When a CoMI server generates an internal event, it is appended to the default stream, and the contents of a notification instance is ready to be sent to all CoMI clients which observe the default stream resource.

Reception of generated notification instances is enabled with the CoAP Observe [I-D.ietf-core-observe] function. The client subscribes to the notifications by sending a GET request with an "Observe" option, specifying the /mg/stream resource.
Every time an event is generated, the default stream is cleared, and the generated notification instance is appended to the stream. After appending the instance, the contents of the instance is sent to all observing clients.

Suppose the server generates the event specified with:

```yaml
module example-port {
    ... 
    prefix ep;
    ...
    notification example-port-fault {
        description
            "Event generated if a hardware fault on a line card port is detected";
        leaf port-name {
            type string;
            description "Port name";
        }
        leaf port-fault {
            type string;
            description "Error condition detected";
        }
    }
}
```

The YANG Hash values for this notification are assigned as follows:

```
1eed4674: /ep:example-port-fault
0cec9c71: /ep:example-port-fault/ep:port-name
228d3fa1: /ep:example-port-fault/ep:fault
```

By executing a GET on the /mg/stream resource the client receives the following response:
REQ: GET example.com/mg/stream
   (observe option register)

RES: 2.05 Content (Content-Format: application/cbor)
   {
     "example-port-fault" : {
       "port-name" : "0/4/21",
       "port-fault" : "Open pin 2"
     }
   }

TODO: fix YANG Hash/CFOR encoding example

RES: 2.05 Content (Content-Format: application/cbor)
   {
     1eed4674 : {
       cec9c71 : "0/4/21",
       228d3fa1 : "Open pin 2"
     }
   }

In the example, the request returns a success response with the contents of the last generated event. Consecutively the server will regularly notify the client when a new event is generated.

To check that the client is still alive, the server MUST send confirmable notifications once in a while. When the client does not confirm the notification from the server, the server will remove the client from the list of observers [I-D.ietf-core-observe].

In the registration request, the client MAY include a "Response-To-Uri-Host" and optionally "Response-To-Uri-Port" option as defined in [I-D.becker-core-coap-sms-gprs]. In this case, the observations SHOULD be sent to the address and port indicated in these options. This can be useful when the client wants the managed device to send the trap information to a multicast address.

4.4. Use of Block

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of text need to be transported the datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [I-D.ietf-core-block] allows the transport of the total payload in individual blocks of which the size can be adapted to the
underlying fragment sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

The block size is specified as exponents of the power 2. The SZX exponent value can have 7 values ranging from 0 to 6 with associated block sizes given by \(2^{(SZX+4)}\); for example SZX=0 specifies block size 16, and SZX=3 specifies block size 128.

The block number of the block to transmit can be specified. There are two block options: Block1 option for the request payload transported with PUT, POST or PATCH, and the block2 option for the response payload with GET. Block1 and block2 can be combined. Examples showing the use of block option in conjunction with observer options are provided in [I-D.ietf-core-block].

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

4.5. Resource Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.mg" [RFC6690]. Upon success, the return payload will contain the root resource of the management data. It is up to the implementation to choose its root resource, but it is recommended that the value "/mg" is used, where possible. The example below shows the discovery of the presence and location of management data.

REQ: GET /.well-known/core?rt=core.mg

RES: 2.05 Content </mg>; rt="core.mg"

Management objects MAY be discovered with the standard CoAP resource discovery. The implementation can add the hash values of the object identifiers to "/.well-known/core" with rt="core.mg.data". The available objects identified by the hash values can be discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.mg.data". Upon success, the return payload will contain the registered hash values and their location. The example below shows the discovery of the presence and location of management data.
Lists of hash values may become prohibitively long. It is discouraged to provide long lists of objects on discovery. Therefore, it is recommended that details about management objects are discovered following the RESTCONF protocol. The YANG module information is stored in the "ietf-yang-library" module [I-D.ietf-netconf-restconf]. The resource "/mg/mod.uri" is used to retrieve the location of the YANG module library.

Since many constrained servers within a deployment are likely to be similar, the module list can be stored locally on each server, or remotely on a different server.

Local in example.com server:

REQ: GET example.com/mg/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
   "mod.uri" : "example.com/mg/modules"
}

Remote in example-remote-server:

REQ: GET example.com/mg/mod.uri

RES: 2.05 Content (Content-Format: application/cbor)
{
   "moduri" : "example-remote-server.com/mg/group17/modules"
}

Within the YANG module library all information about the module is stored such as: module identifier, identifier hierarchy, grouping, features and revision numbers.

The hash identifier is obtained as specified in Section 5.1. When a collision occurred in the name space of the target server, a rehash is executed as explained in Section 5.2.
4.6. Error Return Codes

The RESTCONF return status codes defined in section 6 of the RESTCONF draft are used in CoMI error responses, except they are converted to CoAP error codes.

TODO: complete RESTCONF to CoAP error code mappings

TODO: assign an error cpde for a rehash-error.
<table>
<thead>
<tr>
<th>RESTCONF Status Line</th>
<th>CoAP Status Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 Continue</td>
<td>none?</td>
</tr>
<tr>
<td>200 OK</td>
<td>2.05</td>
</tr>
<tr>
<td>201 Created</td>
<td>2.01</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>none?</td>
</tr>
<tr>
<td>204 No Content</td>
<td>?</td>
</tr>
<tr>
<td>304 Not Modified</td>
<td>2.03</td>
</tr>
<tr>
<td>400 Bad Request</td>
<td>4.00</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>4.03</td>
</tr>
<tr>
<td>404 Not Found</td>
<td>4.04</td>
</tr>
<tr>
<td>405 Method Not Allowed</td>
<td>4.05</td>
</tr>
<tr>
<td>409 Conflict</td>
<td>none?</td>
</tr>
<tr>
<td>412 Precondition Failed</td>
<td>4.12</td>
</tr>
<tr>
<td>413 Request Entity Too Large</td>
<td>4.13</td>
</tr>
<tr>
<td>414 Request-URI Too Large</td>
<td>4.00</td>
</tr>
<tr>
<td>415 Unsupported Media Type</td>
<td>4.15</td>
</tr>
<tr>
<td>500 Internal Server Error</td>
<td>5.00</td>
</tr>
<tr>
<td>501 Not Implemented</td>
<td>5.01</td>
</tr>
<tr>
<td>503 Service Unavailable</td>
<td>5.03</td>
</tr>
</tbody>
</table>

5. Mapping YANG to CoMI payload

A mapping for the encoding of YANG data in CBOR is necessary for the efficient transport of management data in the CoAP payload. Since object names may be rather long and may occur repeatedly, CoMI allows for association of a given object path identifier string value with an integer, called a "YANG hash".

5.1. YANG Hash Generation

The association between string value and string number is done through a hash algorithm. The 30 least significant bits of the "murmur3" 32-bit hash algorithm are used. This hash algorithm is described online at http://en.wikipedia.org/wiki/MurmurHash. Implementation are available online, including at https://code.google.com/p/smhasher/wiki/MurmurHash. When converting 4 input bytes to a 32-bit integer in the hash algorithm, the Little-Endian convention MUST be used.

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

Prefix values are used on every node.

The prefix values defined in the YANG module containing the data object are used for the path expression. For external modules, this is the value of the 'prefix' sub-statement in the 'import' statement for each external module.

Path expressions for objects which augment data nodes in external modules are calculated in the augmenting module, using the prefix values in the augmenting module.

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

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

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

5.2. Re-Hash Error Procedure

A hash collision occurs if two different path identifier strings have the same hash value. If the server has over 30,000 objects in its YANG modules, then the probability of a collision is 10% or higher. If a hash collision occurs on the server, then the object that is causing the conflict has to be altered, such that the new hash value does not conflict with any value already in use by the server.
In most cases, the hash function is expected to produce unique values for all the objects supported by a constrained device. Given a known set of YANG modules, both server and client can calculate the YANG hashes independently, and offline.

Even though collisions are expected to happen rather rarely, they need to be considered. Collisions can be detected before deployment, if the vendor knows which modules are supported by the server, and hence all YANG hashes can be calculated. Collisions are only an issue when they occur at the same server. The client needs to discover any re-hash mappings on a per server basis.

If the server needs to re-hash any object identifiers, then it MUST create a "rehash-map" entry for all its rehashed objects, as described in the following YANG module.

5.3. ietf-yang-hash YANG Module

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

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

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

<CODE BEGINS> file "ietf-yang-hash@2015-06-06.yang"

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

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

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

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

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

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

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

// RFC Ed.: replace XXXX with actual RFC number and remove this note.
// RFC Ed.: remove this note
// Note: extracted from draft-vanderstok-core-comi-07.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-06-06 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: CoMI Protocol.";
}

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

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

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

list object {
  min-elements 2;
  description
    "Each entry identifies one of the objects involved in the
     hash collision and contains the rehash information for
     that object.";

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

  leaf newhash {
    type uint32;
    mandatory true;
    description
      "The new hash value for this object.";
  }

  leaf pathlen {
    type uint32;
    description
      "The length of the path expression of the object with
      this hash value. This object MUST be included
      for any objects in the rehash entry with the

  }

leaf path {
    type string;
    description
    "The path expression of the object with
    this hash value. This object MUST be included
    for any objects in the rehash entry with the
    same 'module' and 'pathlen' values."
}

5.4. YANG Re-Hash Examples

In this example there are three YANG modules, "foo", "bar", and "bar1".
module foo {
    namespace "http://example.com/ns/foo";
    prefix "f";
    revision 2015-06-07;

    container A {
        list B {
            key name;
            leaf name { type string; }
            leaf col1 { type int32; }
            leaf counter1 { type uint32; }
        }
    }
}

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

    leaf bar { type string; }
}

module bar1 {
    namespace "http://example.com/ns/bar1";
    prefix "b1";
    import foo { prefix f; }
    revision 2015-06-07;

    augment /f:A/f:B {
        leaf bar1 { type string; }
    }
}

This set of 3 YANG modules containing a total of 7 objects produces the following object list. Note that actual hash values are not shown, since these modules do not actually cause the YANG Hash clashes described in the examples.
foo:
  container /f:A   h1
  list /f:A/f:B    h2
  leaf /f:A/f:B/f:name  h3
  leaf /f:A/f:B/f:col1  h4
  leaf /f:A/f:B/f:counter1  h5

bar:
  leaf /b:bar       h6

bar1:
  leaf /f:A/f:B/b1:bar1  h7

5.4.1. Multiple Modules

In this example, assume that the following 3 objects produce the same hash value, so 'h3', 'h6', and 'h7' have the same value (e.g. '1234'):

The client might retrieve the container "/f:A" which could cause its sub-nodes to be returned. Instead, the server will return a message with the resource type "core.mg.", representing the "yang-hash" data structure.
REQ: GET example.com/mg/h1

RES: 4.00 "Bad Request" (Content-Format: application/cbor)
{
   "ietf-yang-hash:yang-hash" : {
      "rehash" : [
      {
         "hash" : 1234,
         "object" : [
            {
               "module" : "foo",
               "newhash" : 5678
            },
            {
               "module" : "bar",
               "newhash" : 3579
            },
            {
               "module" : "bar1",
               "newhash" : 8182
            }
         ]
      }]
   }
}

5.4.2. Same Module

In this example, assume that the following 4 objects produce the same hash value, so ‘h3’, ‘h5’, ‘h6’, and ‘h7’ all have the same value (e.g. ‘1234’):

The client might retrieve the list "/f:A/f:B" which would cause its sub-nodes to be returned. Instead, the server will return a message with the resource type "core.mg.yanh-hash", representing the "yang-hash" data structure. Note that the "pathlen" field is not needed for the ‘h6’ and ‘h7’ objects.
REQ: GET example.com/mg/h2?keys="entry1"

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

```
{"ietf-yang-hash:yang-hash": {
"rehash": [

  "hash": 1234,
  "object": [

    { "module": "foo",
      "newhash": 5678,
      "pathlen": 15
    },

    { "module": "foo",
      "newhash": 7863,
      "pathlen": 19
    },

    { "module": "bar",
      "newhash": 3579
    },

    { "module": "bar1",
      "newhash": 8182
    }
  ]
}
}
```

5.4.3. Same Module and Same Path Length

In this example, assume that the following 5 objects produce the same hash value, so 'h3', 'h4', 'h5', 'h6', and 'h7' all have the same value (e.g. '1234'):

The client might retrieve the list "/f:A/f:B" which would cause its sub-nodes to be returned. Instead, the server will return a message with the resource type "core.mg.yang-hash", representing the "yang-hash" data structure. The "path" leaf is included 2 entries because the "module" and "pathlen" values are the same for the objects.
REQ: GET example.com/mg/h2?keys="entry2"

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

```json
{
    "ietf-yang-hash:yang-hash": {
        "rehash": [
            {
                "hash": 1234,
                "object": [
                    {
                        "module": "foo",
                        "newhash": 5678,
                        "pathlen": 15,
                        "path": "/f:A/f:B/f:name"
                    },
                    {
                        "module": "foo",
                        "newhash": 7863,
                        "pathlen": 15,
                        "path": "/f:A/f:B/f:col1"
                    },
                    {
                        "module": "foo",
                        "newhash": 9172,
                        "pathlen": 19
                    },
                    {
                        "module": "bar",
                        "newhash": 3579
                    },
                    {
                        "module": "bar1",
                        "newhash": 8182
                    }
                ]
            }
        ]
    }
}
```

5.5. YANG Hash in URL

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

The hash H is represented as a 30-bit integer, divided into five 6-bit integers as follows:
B1 = (H & 0x3f000000) >> 24
B2 = (H & 0xfc0000) >> 18
B3 = (H & 0x03f000) >> 12
B4 = (H & 0x000fc0) >> 6
B5 = H & 0x00003f

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

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

6. Mapping YANG to CBOR

6.1. High level encoding

When encoding YANG variables in CBOR, the CBOR encodings entry is a map. The key is the YANG hash of entry variable, whereas the value contains its value.

For encoding of the variable values, a CBOR datatype is used. Section 6.2 provides the mapping between YANG datatypes and CBOR datatypes.

6.2. Conversion from YANG datatypes to CBOR datatypes

Table 1 defines the mapping between YANG datatypes and CBOR datatypes.

Elements of types not in this table, and of which the type cannot be inferred from a type in this table, are ignored in the CBOR encoding by default. Examples include the "description" and "key" elements. However, conversion rules for some elements to CBOR MAY be defined elsewhere.

<table>
<thead>
<tr>
<th>YANG type</th>
<th>CBOR type</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8, int16, int32, int64, uint16, uint32, uint64, decimal64</td>
<td>unsigned int (major type 0) or negative int (major type 1)</td>
<td>The CBOR integer type depends on the sign of the actual value.</td>
</tr>
<tr>
<td>boolean</td>
<td>either &quot;true&quot; (major type 7,</td>
<td></td>
</tr>
<tr>
<td>Data Type</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>simple value 21)</td>
<td>or &quot;false&quot; (major type 7, simple value 20)</td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>text string (major type 3)</td>
<td></td>
</tr>
<tr>
<td>enumeration</td>
<td>unsigned int (major type 0)</td>
<td></td>
</tr>
<tr>
<td>bits</td>
<td>array of text strings</td>
<td>Each text string contains the name of a bit value that is set.</td>
</tr>
<tr>
<td>binary</td>
<td>byte string (major type 2)</td>
<td></td>
</tr>
<tr>
<td>empty</td>
<td>null (major type 7, simple value 22)</td>
<td>TBD: This MAY not be applicable to true MIBs, as SNMP may not support empty variables...</td>
</tr>
<tr>
<td>union</td>
<td></td>
<td>Similar to the JSON transcription from [I-D.ietf-netmod-yang-json], the elements in a union MUST be determined using the procedure specified in section 9.12 of [RFC6020].</td>
</tr>
<tr>
<td>leaf-list</td>
<td>array (major type 4)</td>
<td>The array is encapsulated in the map associated with the YANG variable.</td>
</tr>
<tr>
<td>list</td>
<td>array (major type 4) of maps (major type 5)</td>
<td>Each array element contains a map of associated YANG hash - value pairs.</td>
</tr>
<tr>
<td>container</td>
<td>map (major type 5)</td>
<td>The map contains YANG hash - value pairs corresponding to the elements in the container.</td>
</tr>
<tr>
<td>smiv2:oid</td>
<td>array of integers</td>
<td>Each integer contains an element of the OID, the first integer in the array corresponds to the most left element in the OID.</td>
</tr>
</tbody>
</table>
Table 1: Conversion of YANG datatypes to CBOR

7. Error Handling

In case a request is received which cannot be processed properly, the managed entity MUST return an error message. This error message MUST contain a CoAP 4.xx or 5.xx response code, and SHOULD include additional information in the payload.

Such an error message payload is encoded in CBOR, using the following structure:

```plaintext
errorMsg : ErrorMsg;

*ErrorMsg {
    errorCode  : uint;
    ?errorText : tstr;
}
```

The variable "errorCode" has one of the values from the table below, and the OPTIONAL "errorText" field contains a human readable explanation of the error.

<table>
<thead>
<tr>
<th>CoMI Error Code</th>
<th>CoAP Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.00</td>
<td>General error</td>
</tr>
<tr>
<td>1</td>
<td>4.00</td>
<td>Malformed CBOR data</td>
</tr>
<tr>
<td>2</td>
<td>4.00</td>
<td>Incorrect CBOR datatype</td>
</tr>
<tr>
<td>3</td>
<td>4.00</td>
<td>Unknown MIB variable</td>
</tr>
<tr>
<td>4</td>
<td>4.00</td>
<td>Unknown conversion table</td>
</tr>
<tr>
<td>5</td>
<td>4.05</td>
<td>Attempt to write read-only variable</td>
</tr>
<tr>
<td>0..2</td>
<td>5.01</td>
<td>Access exceptions</td>
</tr>
<tr>
<td>0..18</td>
<td>5.00</td>
<td>SMI error status</td>
</tr>
</tbody>
</table>
The CoAP error code 5.01 is associated with the exceptions defined in [RFC3416] and CoAP error code 5.00 is associated with the error-status defined in [RFC3416].

8. Security Considerations

For secure network management, it is important to restrict access to MIB variables only to authorised parties. This requires integrity protection of both requests and responses, and depending on the application encryption.

CoMI re-uses the security mechanisms already available to CoAP as much as possible. This includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorisation mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorisation may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However some adaptations may still be required, to cater for CoMI’s specific requirements.

9. IANA Considerations

'rt="core.mg.data"' needs registration with IANA.

'rt="core.mg.moduri"' needs registration with IANA.

'rt="core.mg.modset"' needs registration with IANA.

'rt="core.mg.yang-hash"' needs registration with IANA.

'rt="core.mg.yang-stream"' needs registration with IANA.

Content types to be registered:

  o application/comi+cbor
10. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification and specified CBOR encoding and use of hashes. Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR. The draft has benefited from comments (alphabetical order) by Dee Denteneer, Esko Dijk, Michael van Hartskamp, Zach Shelby, Michel Veillette, Michael Verschoor, and Thomas Watteyne. The CBOR encoding borrows extensively from Ladislav Lhotka’s description on conversion from YANG to JSON.

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

Juergen Schoenwaelder and Anuj Sehgal were partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

11. Changelog

Changes from version 00 to version 01

  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
Changes from version 02 to version 03
- Added security considerations

Changes from version 03 to version 04
- Added design considerations section
- Extended comparison of management protocols in introduction
- Added automatic generation of CBOR tables
- Moved lowpan table to Appendix

Changes from version 04 to version 05
- Merged SNMP access with RESTCONF access to management objects in small devices
- Added CoMI architecture section
- Added RESTCONF NETMOD description
- Rewrote section 5 with YANG examples
- Added server and payload size appendix
- Removed Appendix C for now. It will be replaced with a YANG example.

Changes from version 04 to version 05
- Extended examples with hash representation
- Added keys query parameter text
- Added select query parameter text
- Better separation between specification and instance
- Section on discovery updated
- Text on rehashing introduced
- Elaborated SMI MIB example
Yang library use described
use of BigEndian/LittleEndian in Hash generation specified

Changes from version 05 to version 06
Hash values in payload as hexadecimal and in URL in base64 numbers
Streamlined CoMI architecture text
Added select query parameter text
Data editing optional
Text on Notify added
Text on rehashing improved with example

Changes from version 06 to version 07
reduced payload size by removing JSON hierarchy
changed rehash handling to support small clients
added LWM2M comparison
Notification handling as specified in YANG
Added Patch function
Rehashing completely reviewed
Discover type of YANG name encoding
Added new resource types
Read-only servers introduced
Multiple updates explained

12. References
12.1. Normative References

12.2. Informative References


Appendix A. Payload and Server sizes

This section provides information on code sizes and payload sizes for a set of management servers. Approximate code sizes are:
<table>
<thead>
<tr>
<th>Code</th>
<th>processor</th>
<th>Text</th>
<th>Data</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observe agent</td>
<td>erbium</td>
<td>800</td>
<td>n/a</td>
<td>[Erbium]</td>
</tr>
<tr>
<td>CoAP server</td>
<td>MSP430</td>
<td>1K</td>
<td>6</td>
<td>[openwsn]</td>
</tr>
<tr>
<td>SNMP server</td>
<td>ATmega128</td>
<td>9K</td>
<td>700</td>
<td>[management]</td>
</tr>
<tr>
<td>Secure SNMP</td>
<td>ATmega128</td>
<td>30K</td>
<td>1.5K</td>
<td>[management]</td>
</tr>
<tr>
<td>DTLS server</td>
<td>ATmega128</td>
<td>37K</td>
<td>2K</td>
<td>[management]</td>
</tr>
<tr>
<td>NETCONF</td>
<td>ATmega128</td>
<td>23K</td>
<td>627</td>
<td>[management]</td>
</tr>
<tr>
<td>JSON parser</td>
<td>CC2538</td>
<td>4.6K</td>
<td>8</td>
<td>[dcaf]</td>
</tr>
<tr>
<td>CBOR parser</td>
<td>CC2538</td>
<td>1.5K</td>
<td>2.6K</td>
<td>[dcaf]</td>
</tr>
<tr>
<td>DTLS server</td>
<td>ARM7</td>
<td>15K</td>
<td>4</td>
<td>[I-D.ietf-lwig-coap]</td>
</tr>
<tr>
<td>DTLS server</td>
<td>MSP430</td>
<td>15K</td>
<td>4</td>
<td>[DTLS-size]</td>
</tr>
<tr>
<td>Certificate</td>
<td>MSP430</td>
<td>23K</td>
<td></td>
<td>[DTLS-size]</td>
</tr>
<tr>
<td>Crypto</td>
<td>MSP430</td>
<td>2-8K</td>
<td></td>
<td>[DTLS-size]</td>
</tr>
</tbody>
</table>

Thomas says that the size of the CoAP server is rather arbitrary, as its size depends mostly on the implementation of the underlying library modules and interfaces.

Payload sizes are compared for the following request payloads, where each attribute value is null (N.B. these sizes are educated guesses, will be replaced with generated data). The identifier are assumed to be a string representation of the OID. Sizes for SysUpTime differ due to preambles of payload. "CBOR opt" stands for CBOR payload where the strings are replaced by table numbers.
### Appendix B. Notational Convention for CBOR data

To express CBOR structures [RFC7049], this document uses the following conventions:

A declaration of a CBOR variable has the form:

```
name : datatype;
```

where "name" is the name of the variable, and "datatype" its CBOR datatype.

The name of the variable has no encoding in the CBOR data.

"datatype" can be a CBOR primitive such as:

- `tstr`: A text string (major type 3)
- `uint`: An unsigned integer (major type 0)
- `map(x,y)`: A map (major type 5), where each first element of a pair is of datatype `x`, and each second element of datatype `y`. A '.' character for either `x` or `y` means that all datatypes for that element are valid.

A datatype can also be a CBOR structure, in which case the variable’s "datatype" field contains the name of the CBOR structure. Such CBOR structure is defined by a character sequence consisting of first its name, then a '{' character, then its subfields and finally a '}' character.

A CBOR structure can be encapsulated in an array, in which case its name in its definition is preceded by a '*' character. Otherwise the structure is just a grouping of fields, but without actual encoding of such grouping.
The name of an optional field is preceded by a ‘?’ character. This means, that the field may be omitted if not required.

Appendix C. comparison with LWM2M

CoMI and LWM2M, both, provide RESTful device management services over CoAP. Differences between the designs are highlighted in this section.

Unlike CoMI, which enables the use of SMIv2 and YANG data models for device management, LWM2M defines a new object resource model. This means that data models need to be redefined in order to use LWM2M. In contrast, CoMI provides access to a large variety of SMIv2 and YANG data modules that can be used immediately.

Objects and resources within CoMI are identified with a YANG hash value, however, each object is described as a link in the CoRE Link Format by LWM2M. This approach by LWM2M can lead to larger complex URIs and more importantly payloads can grow large in size. Using a hash value to represent the objects and resources allows URIs and payloads to be smaller in size, which is important for constrained devices that may not have enough resources to process large messages.

LWM2M encodes payload data in Type-length-value (TLV), JSON or plain text formats. While the TLV encoding is binary and can result in reduced message sizes, JSON and plain text are likely to result in large message sizes when lots of resources are being monitored or configured. Furthermore, CoMI’s use of CBOR gives it an advantage over the LWM2M’s TLV encoding as well since this too is more efficient [citation needed].

CoMI is aligned with RESTCONF for constrained devices and uses YANG data models that have objects containing resources organized in a tree-like structure. On the other hand, LWM2M uses a very flat data model that follows the "object/instance/resource" format, with no possibility to have subresources. Complex data models are, as such, harder to model with LWM2M.

In situations where resources need to be modified, CoMI uses the CoAP PATCH operation when resources are modified partially. However, LWM2M uses the CoAP PUT and POST operations, even when a subset of the resource needs modifications.

Authors’ Addresses
Patch Method for Constrained Application Protocol (CoAP)
draft-vanderstok-core-patch-01

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 a new CoAP method, PATCH, to modify an existing CoAP resource partially.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
1. Introduction

This specification defines the new Constrained Application Protocol (CoAP) [RFC7252] method, PATCH, which is used to apply partial modifications to a resource.

PATCH is also specified for HTTP in [RFC5789]. Most of the motivation for PATCH described in [RFC5789] also applies here.

The PUT method exists to overwrite a resource with completely new contents, and cannot be used to perform partial changes. When using PUT for partial changes, proxies and caches, and even clients and servers, may get confused as to the result of the operation. PATCH was 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 Method

The PATCH 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 can be made by including the If-Match or If-None-Match options in the request (see Section 5.10.8.1 and 5.10.8.2 of [RFC7252]). If the resource could not be created or modified, then an appropriate Error Response Code SHOULD be sent.

The difference between the PUT and PATCH requests is extensively documented in [RFC5789].

PATCH is not safe but idempotent conformant to CoAP PUT specified in [RFC7252], Section 5.8.3.

A PATCH request is idempotent to prevent bad outcomes from collisions between two PATCH requests on the same resource in a similar time frame. These collisions can be detected with the MessageId and the source end-point provided by the CoAP protocol (see section 4.5 of [RFC7252]).

The server MUST apply the entire set of changes atomically and never provide a partially modified representation to a concurrently executed GET request. Given the constrained nature of the servers, most servers will only execute CoAP requests consecutively, thus preventing a concurrent partial overlapping of request modifications. In general, modifications MUST NOT be applied to the server state when an error occurs or only a partial execution is possible. The atomicity requirement holds for all directly affected resources.
A PATCH response can invalidate a cache conformant with the PUT response. Caching behaviour as function of the valid 2.xx response codes for PATCH 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. 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 rather than PUT when the request affects partial updates of a given resource.

2.1. A Simple PATCH Example

The example is taken over from [RFC6902], which specifies a JSON notation for PATCH operations. A resource located at www.example.com/object contains a target JSON document.

```
JSON document original state
{
  "x-coord": 256,
  "y-coord": 45
}

REQ:
PATCH CoAP://www.example.com/object
[
  { "op":"replace","path":"x-coord","value":45}
]
RET:
CoAP 2.04 Changed

JSON document final state
{
  "x-coord": 45,
  "y-coord": 45
}
```
This example illustrates use of a hypothetical PATCH on the /object/ x-coord of the existing resource "object". The 2.04 (Changed) response code is conforms with the CoAP PUT method.

2.2. Response Codes

PATCH for CoAP adopts the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252].

2.3. Option Numbers

PATCH for CoAP adopts the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

3. Error Handling

A PATCH request may fail under certain known conditions. These situations should be dealt with as expressed below.

Malformed PATCH payload: If a server determines that the payload provided with a PATCH request is not properly formatted, it can return a 4.00 (Bad Request) CoAP error. The definition of a malformed payload depends upon the CoAP Content-Format specified with the request.

Unsupported PATCH payload: In case a client sends payload that is inappropriate for the resource identified by the Request-URI, the server can return a 4.15 (Unsupported Content-Format) CoAP error. The server can determine if the payload is supported by checking the CoAP Content-Format specified with the request.

Unprocessable request: This situation occurs when the payload of a PATCH request is determined as valid, i.e. well-formed and supported, however, the server is unable to or incapable of processing the request. The server can return a 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.

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 request cannot be applied to a resource in its current state, or causes the resource to enter an inconsistent state the server can return the 4.09 (Conflict) CoAP response. 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 or lead the device into an inconsistent state if the modifications are made.

Conflicting modification: In situations when a server detects possible conflicting modifications the server can return a 4.nr2 CoAP response code.

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 4.09 (Conflict) CoAP 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 protocol. It is meant to inform protocol and application developers about the security limitations of CoAP PATCH as described in this document. The security consideration of section 15 of [RFC2616], section 11 of [RFC7252], and section 5 of [RFC5789] also apply.

The security considerations for PATCH are nearly identical to the security considerations for PUT ([RFC7252]). The mechanisms used for PUT can be used for PATCH as well.

PATCH is secured following the CoAP recommendations as specified in section 9 of [RFC7252]. When more appropriate security techniques are standardized for CoAP, PATCH can also be secured by those new techniques.
5. IANA Considerations

The entry with name PATCH in the sub-registry, "CoAP Method Codes", is 0.05. The addition will follow the "IETF Review or IESG Approval" procedure as described in [RFC5226].

TODO, definition of CoAP response code 4.09 for addition to the sub-registry of CoAP response codes.

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" [RFC7386].

6. Acknowledgements

Klaus Hartke has pointed out some essential differences between CoAP and HTTP. We are grateful for discussions with Carsten Bormann, Kovatsch Matthias, and Thomas Watteyne.

7. Change log

When published as a RFC, this section needs to be removed.

Version 0 to version 1:

- Changed patch motivation text.
- Removed sub-resource concept.
- Updated cache handling.
- Extended example.
- Update of error handling.

8. References

8.1. Normative References


8.2. Informative References

[I-D.vanderstok-core-comi]

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Anuj Sehgal
Consultant

Email: anuj@iurs.org
Sleepy CoAP Nodes
draft-zotti-core-sleepy-nodes-03

Abstract

6LoWPAN networks rely on application protocols like CoAP to enable RESTful communications in constrained environments. Many of these networks make use of "Sleepy Nodes": battery powered devices that switch off their (radio) interface during most of the time to conserve battery energy. As a result of this, Sleepy Nodes cannot be reached most of the time. This fact prevents using normal communication patterns as specified in the CoRE group, since the server-model is not applicable to these devices. This document discusses and specifies an architecture to support Sleepy Nodes such as battery-powered sensors in 6LoWPAN networks with the goal of guiding and stimulating the discussion on Sleepy Nodes support for CoAP in the CoRE WG.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 7, 2016.
1. Introduction

6LoWPAN networks rely on application protocols such as CoAP to enable RESTful communications in constrained environments. Many of these networks feature "Sleepy Nodes": battery-powered nodes which switch on/off their communication interface to conserve battery energy. As a result of this, Sleepy Nodes cannot be reached most of the time. This fact prevents using normal communication patterns as specified by the CoRE group, since the server model is clearly not applicable to the most energy constrained devices.

This document discusses and specifies an architecture to support Sleepy Nodes such as battery-powered sensors in 6LoWPAN networks. The proposed solution makes use of a Proxy Node to which a Sleepy Node delegates part of its communication tasks while it is not accessible in the 6LoWPAN network. Direct interactions between Sleepy Nodes and non-Sleepy Nodes are only possible, when the Sleepy Node initiates the communication.

Earlier related documents treating the Sleepy node subject are the CoRE mirror server [I-D.vial-core-mirror-server] and the Publish-Subscribe in the Constrained Application Protocol (CoAP) [I-D.koster-core-coap-pubsub]. Both documents describe the interfaces to the proxy accompanying the Sleepy node. Both make use of the observe option discussed in [I-D.ietf-core-observe]. This document describes the roles of the nodes communicating with the Sleepy node and/or its proxy. As such it contributes to understanding how well the other proposals support the operation of the Sleepy nodes in a building control context.

The issues that need to be addressed to provide support for Sleepy Nodes in 6LoWPAN networks are summarized in Section 1.1. Section 2 provides a set of use case descriptions that introduce communication patterns to be used in home and building control scenarios. Section 3, Section 4, Section 5, and Section 6 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 an event message
to another node, or to a group of nodes. After sending this message, the radio and CPU are powered off again, and the Sleepy Node sleeps until the next external event or until a predefined time period has passed. The main problems when introducing Sleepy Nodes into a 6LoWPAN network are as follows:

Problem 1: How to contact a Sleepy Node that has its radio turned off most of the time for:
- Writing configuration settings.
- Reading out sensor data, settings or log data.
- Configuring additional event destination nodes or node groups.

Problem 2: How to discover a Sleepy Node and its services, while the node is asleep:
- Direct node discovery (CoAP GET /.well-known/core as defined in [RFC7252]) does not find the node with high probability.
- Mechanisms may be needed to provide, as the result of node discovery, the IP address of a Proxy instead of the IP address of the node directly.

Problem 3: How a Sleepy Node can convey data to a node or groups of nodes, with good reliability and minimal energy consumption.

1.2. Assumptions

The solution architecture specified here assumes that a Sleepy Node has enough energy to perform bidirectional communication during its normal operational state. This solution may be applicable also to extreme low-power devices such as solar powered sensors as long as they have enough energy to perform commissioning and the initial registration steps. These installation operations may require, in some cases, an additional source of power. Since a Sleepy Node is unreachable for relatively long periods of times, the data exchanges in the interaction model are always initiated by a Sleepy Node when its sleep period ends.

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].
This document assumes readers are familiar with the terms and concepts discussed in [RFC7252],[RFC5988],
[I-D.ietf-core-resource-directory],
[I-D.ietf-core-interfaces],[I-D.ietf-core-observe] and
[I-D.vial-core-mirror-server].

In addition, this document makes use of the following additional terminology:

Sleepy Node: a battery-powered node which does the on/off switching of its communication interface with the purpose of conserving battery energy

Sleeping/Asleep: A Sleepy Node being in a "sleeping state" i.e. its network interface is switched off and a Sleepy Node is not able to send or receive messages.

Awake/Not Sleeping: A Sleepy Node being in an "awake state" i.e. its network interface is switched on and the Sleepy Node is able to send or receive messages.

Wake up reporting duration: the duration between a wake up from a Sleepy Node and the next wake up and report of the same Node.

Proxy: any node that is configured to, or selected to, perform communication tasks on behalf of one or more Sleepy Nodes.

Regular Node: any node in the network which is not a Proxy or a Sleepy Node.

1.4. Acronyms

This Internet-Draft contains the following acronyms:

DTLS: Datagram Transport Layer Security
EP: Endpoint
MC: Multicast
RD: Resource Directory

2. Use cases and architecture

To describe the application viewpoint of the solution, we introduce some example scenarios for the various interactions shown in Figure 1. The figure assigns the following roles taken up by a regular node:
Reading Node: any regular node that reads information from the Sleepy Node.

Configuring Node: any regular node that writes information/configuration into Sleepy Node(s). Examples of configuration are new thresholds for a sensor or a new value for the wake-up cycle time.

Discovering Node: any regular node that performs discovery of the nodes in a network, including Sleepy Nodes.

Destination Node: any regular node or node in a group that receives a message that is generated by the Sleepy Node.

Server Node: an optional server that the Sleepy Node knows about, or is told about, which is used to fetch information/configuration/firmware updates/etc.

Discovery Server: an optional server that enables nodes to discover all the devices in the network, including Sleepy Nodes, and query their capabilities. For example, a Resource Directory server as defined in [I-D.ietf-core-resource-directory] or a DNS-SD server as defined in [RFC6763]. For the rest of this document the discovery server is a Resource Directory. Specifically, the functionalities of the Resource Directory related to the architecture presented in this Internet-Draft are described in more details in Section 3.

The term delegated resource is to indicate the copy at the Proxy of a resource present in the Sleepy Node.

2.1. Node interactions and use cases
The interactions visualized in the figure are discussed and motivated with their use cases:

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 notifications of events to destination Node(s) (directly or via Proxy); for example:

- A battery-powered sensor sends a notification with "battery low" event directly to a designated Destination Node (Direct REPORT).
- A battery-powered occupancy sensor detects an event "people present", switches on the radio and multicast a "ON" command to a group of lights (Direct REPORT).

- 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.

- 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.

- 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.

- A 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.

- 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).

- 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

- 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.

- 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).

- 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="corerp",
</dev/mdl>;rt="ipso.dev.mdl";if="corerp",
</dev/n>;rt="ipso.dev.n";if="corep",
</sen/temp>;rt="ucum.Cel";if="cores"
```

3. 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 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 4.2), 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.

4. Synchronize interface

The functions of the synchronize interface implemented by the Proxy are described in this section.

4.1. 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 3.

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.
4.2. Registration at a Proxy

Once a Sleepy Node has discovered a Proxy by means of one of the procedures described in Section 4.1, 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 3. 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. If present, the End-Point Type parameter SHOULD also be mapped as a "rt" attribute.
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 "</dev..." ------------------------> |
|                                                 |-- 2.01 Created Location: /sp/0 ---------------|

Req: POST coap://sp.example.org/sp?ep=0224e8ffee925dcf&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="0224e8ffe925dcf";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 "</sp/0..." ------------------->   |
    |                                                 |
    |                                                 |
    | <-- 2.01 Created Location: /rd/6534 ----------  |
    |                                                 |
Req: POST coap://rd.example.org/rd?ep=0224e8ffe925dcf&rt=sensor
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

4.3. 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 4.5 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.

4.4. 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.
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.

4.5. 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 4.4.

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.

4.6. 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 4.4.

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 4.5.

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.
5. Delegate Interface

This section details the functions belonging to the delegate interface.

5.1. 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 sleeping node might appear duplicated. EPs SHOULD employ 2-step resource discovery by looking up sleeping 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 sleeping 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 0224e8ffe925dcf is registered on two Proxies.
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 sleeping node with ep=0224e8ffe925dcf.

5.2.  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. The mechanism according to which the Proxy forwards the event to Destination Endpoints (REPORT B) may be linked to a specific protocol (for example: publish/subscribe as in MQTT). A client interested in
the events related with a specific resource may send a CoAP GET to the Proxy, to obtain the last published state. If a Reading node is interested in receiving updates whenever the Sleepy Node reports new event to its Proxy, it can use 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 Endpoint observes the changes of temperature through the Proxy.
5.3. 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.
5.4. 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 5.3) 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.

6. Direct Interface

This section details the functions belonging to the direct interface.

6.1. 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

6.2.  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.

7.  Realization with PubSub server

The registration and discovery of the PubSub broker [I-D.koster-core-coap-pubsub] is covered to the same extent as discussed in this document. Not covered is the direct interaction between Sleepy node and destination nodes as described in Section 6.1. The support from a server node to initialize resources or other information also represents an addition to PubSub broker.

In addition to the continuous updates provided by the PubSub broker, the ad-hoc query of values, the maintenance of operational parameters, the provision of direct update from Sleepy node to a node, the reliability aspects of the update, and the concept of groups are equally important topics that need consideration.

8.  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.

9.  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).

10. 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.

11. 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.

12. References

12.1. Normative References


12.2. Informative References

[I-D.ietf-core-interfaces]

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

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

[I-D.koster-core-coap-pubsub]

[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]


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