CoRE
Internet-Draft
Intended status: Informational
Expires: September 3, 2018

C. Amsuess
March 02, 2018

Resource Directory Replication
draft-amsuess-core-rd-replication-01

Abstract

Discovery of endpoints and resources in M2M applications over large networks is enabled by Resource Directories, but no special consideration has been given to how such directories can scale beyond what can be managed by a single device.

This document explores different ways in which Resource Directories can be scaled up from single network to enterprise and global scale. It does not attempt to standardize any of those methods, but only to demonstrate the feasibility of such extensions and to provide terminology and exploratory groundwork for later documents.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 3, 2018.

Table of Contents

1. Introduction 2
2. Terminology 3
3. Goals of upscaling 3
3.1. Large numbers of registrations 3
3.2. Large number of requests 3
3.3. Redundancy 4
4. Approaches 4
4.1. Shared authority 4
4.2. Plain caching 5
4.3. RD-aware caching 6
4.3.1. Potential for improvement 7
4.4. Distinct registration points 7
4.4.1. Redundancy and handover 8
4.4.2. Loops between RDs and proxies 8
5. Proposed RD extensions 9
5.1. Provenance 9
5.2. Lifetime Age 10
6. Example scenarios 11
6.1. Redundant and replicated resource lookup (anycast) . . . 11
6.2. Redundant and replicated resource lookup (distinct registration points) 12
6.2.1. Variation: Large number of registrations, localized queries 15
6.2.2. Variation: Combination with anycast 15
6.3. Anonymous global endpoint lookup 16
7. References 18
7.1. Informative References 18
7.2. URIs 18
Author's Address 18

1. Introduction

[See abstract for now.]

This document is being developed in a git based workflow. Please see https://github.com/chrysn/resource-directory-replication [1] for more details and easy ways to contribute.

2. Terminology

This document assumes familiarity with [RFC7252] and [I-D.ietf-core-resource-directory] and uses terminology from those documents.

Examples in which URI paths like "/rd" or "/rd-lookup/res" are used assume that those URIs have been obtained before by an RD Discovery process; these paths are only examples, and no implementation should make assumptions based on the literal paths.

3. Goals of upscaling

The following sections outline different reasons why a Resource Directory should be scaled beyond a singe device. Not all of them will necessarily apply to all use cases, and not all solution approaches might be suitable for all goals.

3.1. Large numbers of registrations

Even at 1kB of link data per registration, modern server hardware can easily keep the data of millions of registrations in RAM simultaneously. Thus, the mere size of registration data is not expected to be a factor that requires scaling to multiple nodes.

The traffic produced when millions of nodes with the default 24h lifetime amounts to dozens of exchanges per second, which is doable with equal ease at central network equipment.

However, if the directory has additional interaction with its registered nodes, for example because it provides proxying to registered endpoints, resources like file descriptors can be exhausted earlier, and the traffic load on the registration server grows with the traffic it is proxying for the endpoint.

3.2. Large number of requests

Not all approaches to constrained restful communication use the Resource Directory only in the setup stage; some are might also utilize a Resource Directory in more day-to-day operation.

[TODO: get some numbers on how many requests a single RD can deal with.]

3.3. Redundancy

With the RD as a central part of CoRE infrastructures, outages can affect a large number of users.

A decentralized RD should be able to deal both with scheduled downtimes of hosts as well as unexpected outages of hosts or parts of the network, especially with network splits between the individual parts of the directory.

4. Approaches

In this section, two independent chains of approaches are presented. The "shared authority" approach (using anycast or DNS aliases), and proxy-based caching (in stages from using generic proxies to RD replication that only bears little resemblance to proxies).

In the remainder of this document, the term "proxy" always refers to a device which a client can access as if it were a resource directory, and forwards the request to an actual RD.

Elements from those chains can be mixed.

4.1. Shared authority

With this approach, a single host and port (or "authority" component in the generic URI syntax) is used for all interactions with the RD.

This can be implemented using a host name pointing to different IP addresses simultaneously or depending on the requester’s location, using IP anycast addresses or both.

From the client’s or proxy’s point of view, all interaction happens with same Origin Server.

In this setup, the replication is hidden from the REST interactions, and takes place inside the RD server implementation or its database backend.

Compared to the other approaches, this is more complex to set up when it involves managing anycast addresses: Running an IPv4 anycast network on Internet scale requires running an Autonomous System. In either variation, all server instances are tightly coupled; they need shared administration and probably need to run the same software.

The replication characteristics are laregly inherited from the underlying backend.

As registering endpoints only store the URI constructed from the Location-Path option to their registration request, registration updates can end up at any instance of the server, though they are likely to reach the same one as before most of the time.

Spontaneous failure of individual nodes can interrupt endpoints' registrations in scenarious that do not use anycast addresses until the unusable addresses have left DNS caches.

4.2. Plain caching

Caching reverse proxies that are not particularly aware of a Resource Directory can be used to mitigate the effect of large numbers of requests on a single RD server. In this approach, there exists a single central RD server instance, but proxies are placed in front of it to reduce its load.

Caching is applicable only to the lookup interfaces; the POST request used in registration and renewal are not cacheable.

A prerequisite for successful caching is that fresh copies exist in the cache; this is likely to happen only if there are many alike requests to the Resource Directory. The proxy can than serve cached copies, and might find it advantageous to observe frequent queries.

The simplest way to set up such proxying is to have the proxies forward all requests to the central RD and to advertise only the proxies' addresses.

Due to the discovery process of the RD, operators can also limit the proxies to the lookup interfaces and advertise the central server for registration purposes. A sample exchange between a node and its 6LoWPAN border router could be:

```
Req: GET coap://[fe80::1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
<coap://central-rd.example.com/rd>;rt="core.rd",
<coap://europe3.proxy.rd.example.com/rd-lookup/res>;rt="core.rd-lookup-res",
<coap://europe3.proxy.rd.example.com/rd-lookup/ep>;rt="core.rd-lookup-ep"
```

Special care should be taken when a reverse proxy is not accessed by the client under the same address as the origin server, as relative references change their meaning when served from there. This can be ignored completely on the resource lookup interface (as long as the provenance extension is not used); ignoring it on the endpoint lookup interface gives the client "wrong" results, though that is likely to only matter to applications that use both the lookup and the

registration interface, like Commissioning Tools could do. Proxies can be configured to do content transcoding (cf. [RFC8075] Section 6.5.2) to preserve the lookup responses' original meanings.

This approach does not help at all with large numbers of registrations. It can mitigate issues with large numbers of lookup requests, provided that many identical requests arrive at the proxy. The effect on the redundancy goal is negligible: The proxy can provide lookup results only for as long as the cache is fresh during a central server outage, which is 60 seconds unless the RD server says otherwise.

This approach can be run with off-the-shelf RD servers and proxies. The only configuration required is for the proxy to have a forwarding address, and for the RD (or its announcer) tho know which lookup addresses to advertise.

4.3. RD-aware caching

Similar to the above, specialized proxies can be employed that are aware that their target is an RD lookup address.

The "plain caching" approach is limited in that it requires a small set of lookups to be frequently performed. A proxy that is aware that the address it is forwarding to is of the Resource Type "core.rd-lookup-*" can utilize knowledge of how an RD works to serve more specialized requests as well from fresh generic content.

For example, assume that the proxy frequently receives requests of the shape

```
Req: GET /rd-lookup/res?rt=core.s&rt=ex.temperature&ex.building=8341&title=X
```

for arbitrary values of X. Then it can use the following request to keep a fresh cache:

```
Req: GET coap://rd.example.com/rd-lookup/res?rt=core.s&rt=ex.temperature
    &ex.building=8341
Observe: 1
```

and from that serve filtered responses to individual requests.

This method shares the advantages of plain caching, with reduced limitations but requiring specialized proxying software. The software does not necessarily need more configuration: A general-purpose proxy is free to explore the origin server's ".well-known/core" information, and can decide to enable RD optimizations after

discovering that the frequently accesses resources are of resource type "core.rd-lookup-*".

4.3.1. Potential for improvement

Observing a large lookup result is relatively inefficient as the complete document needs to be transferred when a change happens. Serializations of web links that are suitable for expressing small deltas are expected to be developed for PATCH operations on registration resources. If those formats are compatible with observation, they can be applied directly. Otherwise, the proxy can try to establish a "push" dynamic link ([I-D.ietf-core-dynlink]) to receive continuous PATCH updates on its resource.

The applicability of the RD-aware approach is further limited to query parameters of which the proxy knows that they are not subject to lookup filtering on other entities than the queried one. In the example above, were the variable part the "d" attribute (of endpoints, as opposed to the "title" of resources), the proxy could not do the filtering on its own becaus it would not have the required information. Even the above example does not allow for fully accurate replication, as the endpoint _might_ register with a "title" endpoint attribute, even though no such attribute is specified right now. Also, annotating the links in the endpoint lookup with information about which registration they belong to would help the proxy keep all the data around to solve more complex queries. The provenance extension is proposed for that purpose.

In its extreme form, the proxy can observe the complete lookup resources of the Resource Directory. It can then answer all queries on its own based on the continuously fresh state transferred in the observations. That form requires the RD to support the provenance extension.

For such proxies, it can be suitable to configure them to use stale cache values for extended periods of time when the RD becomes intermittently unavailable.

4.4. Distinct registration points

Caching proxies that are aware of RD semantics could be extended to gather information from more than one Resource Directory.

When executing queries, they would consider candidates from all configured upstream servers and report the union of the respective query results. At this stage, it is highly recommended that content transcoding takes place.

With this approach, many distinct registration URIs can be advertised, for example due to geographic proximity.

Unlike the other proxying approaches, this helps with the "large number of registrations" goal. If that number is unmanageable for single devices, proxies need not keep full copies of all the RDs' states but rather send out queries to all of their upstreams, behaving more like the "plain caching" proxies. This multiplies the lookup traffic, but allows for huge numbers of registrations. The problems of "too many lookups" versus "too many registrations" can be traded off against each other if the proxies keep parts of the RDs' states locally at hand while forwarding more exotic requests to all RDs.

4.4.1. Redundancy and handover

This approach also tackles the redundancy goal. When an endpoint registeres at its RD, the RD updates its endpoint and resource lookup results and includes the registration data until further notice (for correct operation, the "Lifetime Age" extension is useful).

If at some point in time that RD server becomes unavailable, the proxies can keep the cached information around. Before the lifetime expires, the endpoint will attempt to renew its registration and find that the RD is unavailable. It will then go through discovery again, find the most recently advertised registration URI or pick another one out of a set and start a new registration there.

If the lookup proxies do not evict the old (and soon-to-time-out) registration when the new one on a different RD with the same endpoint name and domain arrives, at worst there will be the same information twice from two registration resources available for lookup.

4.4.2. Loops between RDs and proxies

In this configuration, it can be tempting to run a Resource Directory and a lookup proxy (aimed at multiple resource directories) on the same host.

[It might be easier to recommend simply using different hosts, at least host names, in those cases, or anything else that allows direct and not publically advertised access to the real RDs' lookups.]

In such a setup, other aggregating lookup proxies must take care to only select locally registered entries. With the current filtering rules, observing the resources "/rd-lookup/ep?href=/*" and "/rd-lookup/res?provenance=/*" crudely provides that kind of filtering.

5. Proposed RD extensions

5.1. Provenance

In order for an RD-aware proxy to serve resource lookup requests that filter on endpoint parameters, the proxy needs a way to tell which endpoint registration submitted that link. That information might also be useful for other purposes.

This introduces a new link attribute "provenance". Its value is a URI reference as described by [RFC3986] Section 4.1. The URI is to be interpreted by the same rules that apply to the "anchor" attribute, namely by resolving the reference relative to the requested document's URI. The attribute should not be repeated, and in presence of multiple attributes, only the last should be considered.

[TODO: If a something link-format-ish comes up during the development of this document which allows setting base-hrefs in-line, evaluate whether it really makes sense to inherit anchor's rules or whether it's better to phrase it in a way that the requested base URI always counts.]

The URI given in the "provenance" attribute describes where the information in the link was obtained from. An aggregator of links can thus declare its sources for each link.

It is recommended that a Resource Directory adds the URI of the registration resource to resource lookups. Thus, if an endpoint registers as

```
Req: POST /rd?ep=node1
Payload:
</sensors/temp>;if="core.s"

Res: 2.01 Created
Location: /reg/1234
```

then a lookup will add a provenance attribute:

```
Req: GET /rd-lookup/res?if=core.s

Res: 2.05 Content
Payload:
<coap://.../sensors/temp>;if="core.s";anchor="coap://...";
    provenance="/reg/1234"
```

This is not an IANA consideration as there is no established registry of link attributes.

By itself, the provenance attribute does not need to be registered in the RD Parameters Registry because it is just another link attribute. If it is desired that provenance information is only shown on request (eg. by RD-aware proxies), a parameter can be introduced there:

- Full name: Link provenance
- short: provenance
- Validity: URI
- Use: Resource lookup only
- Description: If "provenance" or any string starting with "provenance=" is given as one of the ampersand-delimited query arguments, the RD is instructed to add the provenance attribute to all looked up links; otherwise, the RD will not present them. The filtering rules still apply: If there is a "=" sign in the query argument, only links with matching provenance will be reported.

5.2. Lifetime Age

The result of an endpoint lookup as a whole has inhomogenous cache properties that would determine its Max-Age:

- The document can change at any time when a new endpoint registers.
- The document can change at any time when an endpoint deregisters.
- Each record can be expected to not change until its lifetime has expired.

As currently specified, a lookup client has no way to tell where in its lifetime an endpoint is. Therefore, a new link attribute is suggested that allows the RD to share that information:

The new link attribute Lifetime Age (lt-age) is described for use in RD Endpoint Lookups. Valid values are integers from 0 to the lifetime of the registration. The value indicates how many seconds have passed since the endpoint last renewed its registration.

Care has to be taken when replicating this value in caches, as the caching agent might be unaware of the attribute’s semantics and not update it. (This is unlike the Max-Age attribute, which a caching agent needs to understand and reduce accordingly when serving from

the cache). It should therefore only be used with responses that carry the default Max-Age of 60 or less.

Clients that use the lookup interface (especially RD-aware proxies) are free to treat that record and its corresponding resource records as fresh until after the difference of lt and lt-age seconds have passed since the endpoint lookup result was obtained, especially if the origin server has become unavailable.

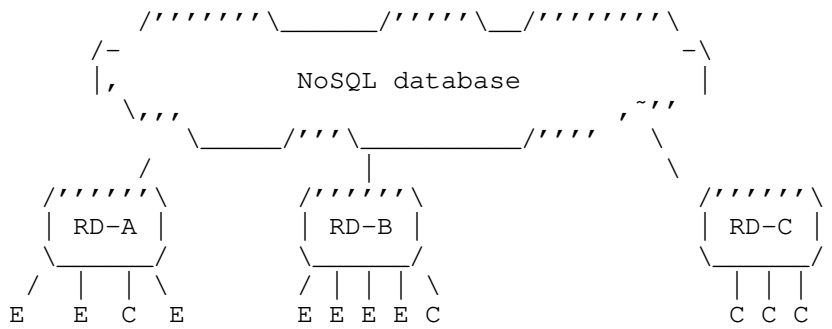
Security considerations: Given that this leaks information about the endpoint's communication patterns, it may be prudent for an RD only to reveal this information on a need-to-know basis.

6. Example scenarios

6.1. Redundant and replicated resource lookup (anycast)

This scenario describes a setup where millions of devices register in a company-wide Resource Directory.

The directory is scaled using the shared authority / anycast approach, and the RD implementation is backed by a NoSQL-style distributed database.



("E" and "C" represent endpoints and lookup clients, respectively)

Both endloints and lookup clients receive the RD address "2001:db8::an1:ca57" is announced to all devices on the network using the RDAO option in IPv6 Neighbor Discovery. Any packages to that addresses are routed by the network to the closest of the three RD instances A, B and C. Discovery invariably looks like this:

```
Req: GET coap://[2001:db8::an1:ca57/.well-known/core?rt=core.rd*

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

An endpoint close to B would therefore register with

```
Req: POST coap://[2001:db8::an1:ca57]/rd?ep=endpoint1&
    d=facility23.eu.example.com
Payload:
</sensors/temp>;if="core.s"

Res: 2.01 Created
Location: /reg/123e4567-e89b-12d3-a456-426655440000
```

Any client could immediately see that the endpoint is registered by issuing

```
Req: GET coap://[2001:db8::an1:ca57]/rd-lookup/ep?ep=endpoint1&
    d=facility23.eu.example.com

Res: 2.05 Content
Payload:
</reg/123e4567-e89b-12d3-a456-426655440000>;ep="endpoint1";
    d="facility23.eu.example.com";con="coap://[2001:db8:23::1]"
```

If at any point in time the RD instance B becomes unavailable, the registering endpoint's renewal requests will be routed to the next available instance, for example A. That instance can update the shared database with renewed lifetime just as B would have done.

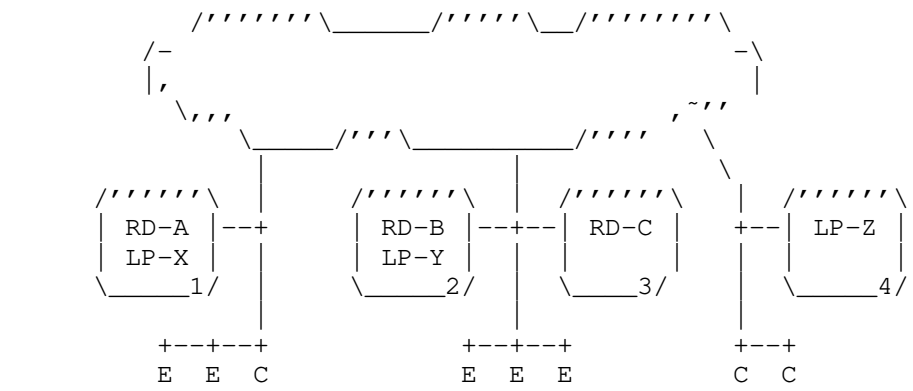
How this performs under a net split depends on the database backend. Registration resources based on UUIDs were chosen in this example because those would allow the system to keep accepting new registrations even in a netsplit situation; the risk of the registration request not being idempotent towards a node that switches sides during such a split is considered acceptable.

6.2. Redundant and replicated resource lookup (distinct registration points)

This scenario takes place in the same environment as the previous one.

Rather than a shared database, distinct registration points are advertised. The advertised registration points are called RD-A to

RD-C; independent of them are lookup proxies LP-X to LP-Z. Some of them run on the same hosts.



The lookup proxies in this scenario are constantly observing the "/rd-lookup/ep?href=/*" and "/rd-lookup/res?provenance=/*" resources of known RDs on other hosts, and might get updated internally with state from a co-hosted RD or observe that using an internal interface. As there is really suitable content format and observation mechanism for those yet, the exchanges are partially described in words here.

RDAO announcements point to the nearest host (whose IP address ends with the numbers of the respective box in the figure), and hosts that do not serve both functions provide lookup as follows:

Req: GET coap://[2001:db8:23::3]/.well-known/core?rt=core.rd*

Res: 2.05 Content
Payload:
</rd>;rt="core.rd",
<coap://[2001:db8:23::2]/rd-lookup/ep>;rt="core.rd-lookup-ep",
<coap://[2001:db8:23::2]/rd-lookup/res>;rt="core.rd-lookup-res"

When a client then registers as

Req: POST coap://[2001:db8:23::3]/rd?ep=endpoint1&
    d=facility23.eu.example.com
Payload:
</sensors/temp>;if="core.s"

Res: 2.01 Created
Location: /reg/42

the RD at 3 sends notifications to the observing lookup proxies X, Y and Z:

```
Res: Patch Result
Add one record: </reg/42>;ep="endpoint1";d="facility23.eu.example.com";
    con="coap://[2001:db8:23::1]";lt-age=0
```

As soon as that is processed, clients can query LP-Z

```
Req: GET coap://[2001:db8:4::4]/rd-lookup/ep?ep=endpoint1&
    d=facility23.eu.example.com

Res: 2.05 Content
Payload:
<coap://[2001:db8:23::3]/reg/42>;ep="endpoint1";
    d="facility23.eu.example.com";con="coap://[2001:db8:23::1]"
```

(Note that lt-age is elided to the client as per the security considerations for that information).

When a net split happens that cuts LP-Z's site off the rest, it keeps that information available until the lt-age runs out.

When RD-C unexpectedly becomes unavailable, endpoint1 fails to renew its registration. It then starts the RD discovery process again, picks the next available RD (this time B) and gets a new registration from that.

RD-B then sends an update to the proxies:

```
Res: Patch Result
Add one record: </reg/11>;ep="endpoint1";d="facility23.eu.example.com";
    con="coap://[2001:db8:23::1]";lt-age=0
```

The proxies remove C's registration "/reg/42" based on the duplicate name and now answer requests like this:

```
Req: GET /rd-lookup/ep?ep=endpoint1&d=facility23.eu.example.com

Res: 2.05 Content
Payload:
<coap://[2001:db8:23::2]/reg/11>;ep="endpoint1";
    d="facility23.eu.example.com";con="coap://[2001:db8:23::1]"

Req: GET /rd-lookup/res?if=core.s&d=facility23.eu.example.com

Res: 2.05 Content
Payload:
<coap://[2001:db8:23::1]/sensors/temp>;if="core.s";
    anchor="coap://[2001:db8:23::1]/sensors/temp";
    provenance="coap://[2001:db8:23:2]/reg/11",
...
```

6.2.1. Variation: Large number of registrations, localized queries

If the lookup proxies are not capable of keeping track of all the registered data, they can opt to forward requests to all the RDs instead. In this example, queries are often localized (queries within a building are often limited to the same building), so LP-Y could decide to only keep two particular observations active to each RD:

- "/rd-lookup/ep?href=/*&d=facility23.eu.example.com"
- "/rd-lookup/res?provenance=/*&d=facility23.eu.example.com"

With those observed, it could still accurately respond to the above queries without calling out to the other RDs.

If a query came in as "/rd-lookup/res?if=core.s", it would still need to forward that query to all RDs to build an overview of all sensors in the network for the requester.

6.2.2. Variation: Combination with anycast

In a variation of this, all the RDs and LPs can use a shared anycast address. They would be then advertised as in the anycast/NoSQL example.

All RDs would need to be configured such that they encode their host name in their path (eg. "/reg/rd-c/42"). Nodes must then have proxy forwarding rules set up such that

- "/rd" is served from the local RD if there is one, or forwarded to any (the closest) RD

- "/reg/*" requests are served if hosted locally, otherwise forwarded to the appropriate RD, or respond with a 5.04 Gateway timeout if that is not available any more

- Lookup request are served from the local lookup proxy, or forwarded to the closest one on RD-only hosts.

Such a setup is easier if all hosts provide both registration and lookup functionality.

6.3. Anonymous global endpoint lookup

This scenario describes a way to provide connectivity into devices in difficult network situations based on identifiers of their cryptographic keys, the KID context identifiers of OSCORE. A global network of untrusted Resource Directory servers is built, and the individual servers provide network relaying services for endpoints that operate behind NAT or firewalls.

It assumes the existance of two other hypothetical mechanisms:

- The RD Parameter named "proxy".

  An endpoint can ask the RD to act as a reverse proxy for it by adding the "proxy" registration parameter; an RD that does proxying disregards the implicit "con" parameter and announces a name of its own instead.

- A URI scheme called "oscore".

  A URI of the form "oscore://VGhh...2aWNl/sensor/temp" refers to a resource "/sensor/temp/" on any OSCORE capable host with which the client has a key established under the KID context given by the base64 string in the authority component.

  To resolve the URI to a concrete protocol and socket, a form of Resource Directory assisted protocol negotiation is used.

RD servers join a global pool of servers using a protocol that is not further described here, but could conceivably be based on distributed hash tables (DHTs).

Endpoints register only with a key derived name, and usually do not provide any links because those would be accessible only to authenticated requesters.

They register at any of a set of preconfigured DNS names for finding a Resource Directory. Those names resolve to any of the currently

active RD servers, where geographic proximity could play a role in the choice of address returned.

When the endpoint discovers the registration URI (for which it uses coap+tcp to make later proxying more stable), the server returns links to its explicit IP address:

```
<coap+tcp://[2001:db8:1:2::3]/rd>;rt="core.rd",
<coap+tcp://[2001:db8:1:2::3]/rd-lookup/ep>;rt="core.rd-lookup-ep"
```

(This avoids conflict when the DNS assignment flips and a different host (on which the registration resource is unknown) is returned. Alternatively, the servers could use a unified scheme of registration resource naming like "/reg/${name}" or a UUID-based scheme.)

The endpoint then registers:

```
Req: POST coap+tcp://[2001:db8:1:2::3]/rd?proxy&ep=VGhhdCdzIHRoZSB\
    LZXlJZENvbnRleHQgdXNlZCB3aXRoIHRoaXMgZGV2aWNl
Payload: empty

Res: 2.01 Created
Location: /reg/123
```

When a client wants to talk to that registered server, its RD discovery process will yield another instance, which it then queries:

```
Req: GET coap://[2001:db8:4:5::6]/rd-lookup/ep?ep=VGhhdCdzIHRoZSBL\
    ZXlJZENvbnRleHQgdXNlZCB3aXRoIHRoaXMgZGV2aWNl
```

The server will look up the given ep name in the backing DHT, and forward the request right to the (precisely: any) RD server that has announced that ep value, which then answers:

```
Res: 2.05 Created
Payload:
<coap+tcp://[2001:db8:1:2::3]/reg/123>;ep="VGhh...2aWNl";
    con="coap://[2001:db8:1:2::3]:10123";
    at="coap+tcp://[2001:db8:1:2::3]:10123"
```

(This particular server uses multiple ports to tell traffic for different endpoints apart; it could just as well use a catch-all DNS record, do name based virtual hosting and announce "con="coap://reg123.server3.example.com" instead.)

The client will then use the discovered address to direct its OSCORE requests to, and the RD server will proxy for it.

Note that while this setup _can_ serve as a generic RD and answer resource requests as well, it is doubtful whether there would be any interest in it given the data becomes public, and is limited by the necessity to have an "ep=" filter in all requests lest the network be flooded with requests.

7. References

7.1. Informative References

[I-D.ietf-core-dynlink]
Shelby, Z., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-04 (work in progress), September 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-13 (work in progress), March 2018.

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

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

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

7.2. URIs

[1] https://github.com/chrysn/resource-directory-replication

Author’s Address

Christian Amsuess
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: christian@amsuess.com

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

P. van der Stok
consultant
K. Hartke
Universitaet Bremen TZI
February 26, 2018

"Pending" Responses for the Constrained Application Protocol (CoAP)
draft-hartke-core-pending-02

Abstract

This document proposes a new type of response for the Constrained Application Protocol (CoAP) called a "Pending" response. A CoAP server can use a Pending response to indicate that it has accepted a request but has not yet started processing it or that processing the request will take longer than a client is typically willing to wait for a response.

Table of Contents

1. Introduction 2
1.1. Terminology 3
2. Pending Responses 3
2.1. Observing Resources 4
3. Security Considerations 4
4. IANA Considerations 5
5. References 5
5.1. Normative References 5
5.2. Informative References 5
Authors' Addresses 6

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a request/response protocol not unlike HTTP. CoAP defines no upper bound for the time between a request and the resulting response. For example, a CoAP-over-UDP server is expected to return an empty Acknowledgement to the client if it cannot provide a response right away, but there is no limit on the time when the server should return the Separate Response.

In particular in the case of requests with long processing times, a CoAP client faces the problem that it cannot easily determine how long it should wait for the response and whether the CoAP server is actually still processing the request. Long processing times occur, for example, when requests need manual intervention to authorize their processing, or when they perform a long sequence of remote actions. An example for this is the "possibly long" authorization request specified in EST-coaps [I-D.vanderstok-ace-coap-est].

This document proposes a new kind of response in CoAP, called a "Pending" response. The semantics of this response are modelled after the HTTP 202 (Accepted) status code [RFC7231]:

> The 202 (Accepted) status code indicates that the request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. [...] The representation sent with this response ought to describe the request's current status and point to (or embed) a status monitor that can provide the user with an estimate of when the request will be fulfilled.

Pending responses are not intended for overload cases, which are better handled by the 5.03 (Service Unavailable) response code.

1.1. Terminology

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

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

2. Pending Responses

A Pending response is denoted by a response code in the 2.xx range and a Content-Format Option that is set to content-format ID TBD1.

A 2.01 (Creation Pending) response in reply to a POST request indicates that the result of processing the request is not available yet, for example, because the server needs more time to process the request than a client is typically willing to wait for a response. The server MAY specify a location using the Location-* options where the result will become available. If the server does not specify a location, the result will become available at the target resource of the POST request. To retrieve the result, the client MAY poll or observe the resource at this location using the GET request method.

A 2.02 (Deletion Pending) response in reply to a DELETE request indicates that the server has accepted the request but the target has not been fully deleted yet.

A 2.04 (Change Pending) response in reply to a POST or PUT request indicates that the server has accepted the request but the result of processing the request is not available yet.

A 2.05 (Content Pending) response in reply to GET request indicates that the target resource exists but a representation of the resource is not available yet. The Max-Age Option indicates after what time a client should retry its GET request to retrieve the representation. The client MAY observe the resource [RFC7641] to get notified when the representation becomes available (see Section 2.1 for details).

The payload of a Pending response MAY be a brief human-readable diagnostic message, explaining the situation, or MUST be absent.

The cacheability of Pending responses is as specified for the respective response code.

2.1. Observing Resources

When a client registers to observe a resource [RFC7641] for which no representation is available yet, the server MAY send one or more 2.05 (Content Pending) notifications before sending the first actual 2.05 (Content) or 2.03 (Valid) notification. The possible resulting sequence of notifications is shown in Figure 1.

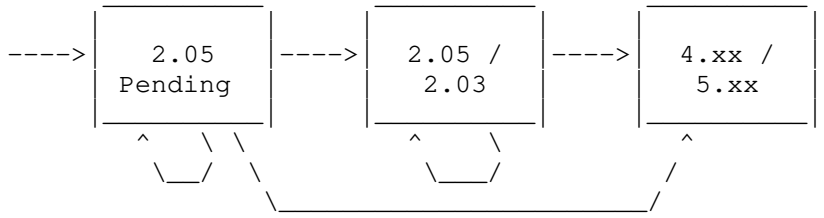


Figure 1: Sequence of Notifications

Unless the server is unwilling to add the client to the list of observers, each 2.05 (Content Pending) notification MUST include an Observe Option with a sequence number as specified in [RFC7641]. Otherwise, the registration request falls back to a normal GET request.

3. Security Considerations

This section analyses the possible threats related to Pending responses. It is meant to inform protocol and application developers about the security limitations of the response code as described in this document.

A Pending response is subject to the same general security considerations as all CoAP responses as described in Section 11 of [RFC7252]. Specifically, the security considerations for the response code are closest to those of the Observe Option as stated in Section 7 of [RFC7641], because the server stores additional state over an extended period.

Pending responses are secured following the recommendations for the existing CoAP response codes as specified in Section 9 of [RFC7252]. When additional security techniques are standardized for CoAP (e.g., based on object security), these are then also available for securing the responses.

4. IANA Considerations

This document adds the content-format used to signal Pending responses to the "CoAP Content-Formats" registry.

Media Type	Content Coding	ID	Reference
-	-	TBD1	[This Document]

New CoAP Content-Formats

TBD1 is taken from the "First Come First Served" range of the "CoAP Content-Formats" registry.

5. References

5.1. Normative References

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

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

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

5.2. Informative References

[I-D.vanderstok-ace-coap-est] Stok, P., Kampanakis, P., Kumar, S., Richardson, M., Furuhed, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-vanderstok-ace-coap-est-04 (work in progress), January 2018.

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

Authors' Addresses

Peter van der Stok
consultant

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

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

Phone: +49-421-218-63905
Email: hartke@tzi.org

ACE
Internet-Draft
Intended status: Standards Track
Expires: September 1, 2018

P. van der Stok
Consultant
P. Kampanakis
Cisco Systems
S. Kumar
Philips Lighting Research
M. Richardson
SSW
M. Furuhed
Nexus Group
S. Raza
RISE SICS
February 28, 2018

EST over secure CoAP (EST-coaps)
draft-ietf-ace-coap-est-00

Abstract

Enrollment over Secure Transport (EST) [RFC7030] is used as a certificate management protocol over HTTPS.

Low-resource devices often use the lightweight Constrained Application Protocol (CoAP) [RFC7252] for message exchanges. This document defines how to transport EST payloads over secure CoAP (EST-coaps). This allows low-resource constrained devices to re-use existing EST functionality. Example low-resource use cases for EST are: secure bootstrapping and certificate enrollment.

Table of Contents

1. Introduction 3
1.1. EST operational differences 3
1.2. Terminology 4
2. Conformance to RFC7925 profiles 4
3. Protocol Design and Layering 5
3.1. Payload format 6
3.2. Message Bindings 6
3.3. CoAP response codes 6
3.4. Message fragmentation 7
3.5. Deployment limits 8
4. Discovery and URI 8
5. DTLS Transport Protocol 10
6. Proxying 11
7. Parameters 12
8. IANA Considerations 12
8.1. Content-Format registry 12
8.2. Resource Type registry 14
9. Security Considerations 15
9.1. proxy considerations 15
9.2. EST server considerations 15
10. Acknowledgements 16
11. Change Log 16
12. References 16
12.1. Normative References 16
12.2. Informative References 17
Appendix A. EST messages to EST-coaps 19
A.1. cacerts 20
A.2. csrattrs 22
A.3. enroll / reenroll 22
A.4. serverkeygen 24
Appendix B. Encoding for server side key generation 26

Appendix C. EST-coaps Block message examples 26
Authors' Addresses 28

1. Introduction

Enrollment over Secure Transport (EST) [RFC7030] is used for authenticated/authorized endpoint certificate enrollment (and optionally key provisioning) through a Certificate Authority (CA) or Registration Authority (RA). This functionality is also needed for low resource devices.

"Classical" EST uses HTTPS and this specification defines a new transport for EST using CoAP. It also profiles the use of EST to a smaller subset.

IPv6 over Low-power Wireless Personal Area Networks (6LoWPANs) [RFC4944] on IEEE 802.15.4 [ieee802.15.4] wireless networks are becoming common in many industry application domains such as lighting controls. Although IEEE 802.15.4 defines how security can be enabled between nodes within a single mesh network, it does not specify the provisioning and management of the keys. Therefore, securing a 6LoWPAN network with devices from multiple manufacturers with different provisioning techniques is often tedious and time consuming. An example use case is the application of Bootstrapping of Remote Secure Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra]. The low resource aspects are detailed for 6tisch in [I-D.ietf-6tisch-minimal-security] and [I-D.ietf-6tisch-dtsecurity-secure-join].

Constrained networks use DTLS [RFC6347], CoAP [RFC7252], and UDP instead of TLS [RFC5246], HTTP [RFC7230] and TCP. EST-coaps replaces the invocations of TLS and HTTP by DTLS and CoAP invocations thus enabling EST for CoAP-based low-resource devices.

Because the relatively large EST messages cannot be readily transported over constrained (6LoWPAN, LLN) wireless networks, this document specifies the use of CoAP Block-Wise Transfer ("Block") [RFC7959] to fragment EST messages at the application layer.

1.1. EST operational differences

Only the differences to EST with respect to operational scenarios are described in this section. EST-coaps server differs from EST server as follows:

- Replacement of TLS by DTLS and HTTP by CoAP, resulting in:

  * DTLS-secured CoAP sessions between EST-coaps client and EST-coaps server.

- Only certificate-based client authentication is supported, which results in:

  * The EST-coaps client does not support HTTP Basic authentication (as described in Section 3.2.3 of [RFC7030]).

  * The EST-coaps client does not support authentication at the application layer (as described in Section 3.2.3 of [RFC7030]).

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].

Many of the concepts in this document are taken over from [RFC7030]. Consequently, much text is directly traceable to [RFC7030]. The same document structure is followed to point out the differences and commonalities between EST and EST-coaps.

2. Conformance to RFC7925 profiles

This section shows how EST-coaps fits into the profiles of low-resource devices as described in [RFC7925].

EST-coaps can transport certificates and private keys. Private keys can be transported as response to a request to a server-side key generation as described in section 4.4 of [RFC7030].

The mandatory cipher suite for DTLS is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 defined in [RFC7251] which is the mandatory-to-implement cipher suite in CoAP. Additionally, the curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. DTLS implementations MUST use the Supported Elliptic Curves and Supported Point Formats Extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090] can be used as an implementation method.

The EST-coaps client MUST be configured with an explicit TA database or at least an implicit TA database from its manufacturer. The authentication of the EST-coaps server by the EST-coaps client is based on Certificate authentication in the DTLS handshake.

The authentication of the EST-coaps client is based on client certificate in the DTLS handshake. This can either be

- o DTLS with a previously issued client certificate (e.g., an existing certificate issued by the EST CA); this could be a common case for simple re-enrollment of clients;

- o DTLS with a previously installed certificate (e.g., manufacturer-installed certificate or a certificate issued by some other party);

3. Protocol Design and Layering

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [RFC7959] to transport CoAP messages in blocks thus avoiding (excessive) 6LoWPAN fragmentation of UDP datagrams. The use of "Block" for the transfer of larger EST messages is specified in Section 3.4. The Figure 1 below shows the layered EST-coaps architecture.

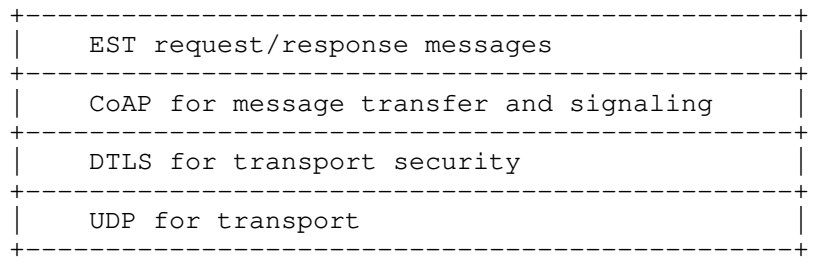


Figure 1: EST-coaps protocol layers

The EST-coaps protocol design follows closely the EST design. The parts supported by EST-coaps are identified by their message types:

- o Simple enroll and reenroll, for CA to sign public client-identity key.

- o CA certificate retrieval, needed to receive the complete set of CA certificates.

- o CSR Attributes request messages, informs the client of the fields to include in generated CSR.

- o Server-side key generation messages, to provide a private client-identity key when the client is too restricted or because of lack of an entropy source. [EDNOTE: Encrypting these keys is important. RFC7030 specifies how the private key can be encrypted with CMS using symmetric or asymmetric keys. Mention how symmetric key can be derived for EST server side key generation from the TLS KEM draft.]

3.1. Payload format

The content-format (media type equivalent) of the CoAP message determines which EST message is transported in the CoAP payload. The media types specified in the HTTP Content-Type header (see section 3.2.2 of [RFC7030]) are in EST-coaps specified by the Content-Format Option (12) of CoAP. The combination of URI path-suffix and content-format used for CoAP MUST map to an allowed combination of path-suffix and media type as defined for EST. The required content-formats for these request and response messages are defined in Section 8. The CoAP response codes are defined in Section 3.3.

EST-coaps is designed for use between low-resource devices using CoAP and hence does not need to send base64-encoded data. Simple binary is more efficient (30% less payload compared to base64) and well supported by CoAP. Therefore, the content formats specification in Section 8 requires the use of binary for all EST-coaps Content-Formats.

3.2. Message Bindings

This section describes the general EST CoAP message characteristics.

It is RECOMMENDED to use CoAP CON messages. This recommendation does not influence the communication efficiency because all EST-coaps messages expect a response.

The Ver, TKL, Token, and Message ID values of the CoAP header are not influenced by EST.

CoAP options are used to convey Uri-Host, Uri-Path, Uri-Port, Content-Format and more in CoAP. The CoAP Options are used to communicate the HTTP fields specified in the EST REST messages.

EST URLs are HTTPS based (https://), in CoAP these will be assumed to be transformed to coaps (coaps://)

Appendix A includes some practical examples of EST messages translated to CoAP.

3.3. CoAP response codes

Section 5.9 of [RFC7252] specifies the mapping of HTTP response codes to CoAP response codes. Every time the HTTP response code 200 is specified in [RFC7030] in response to a GET (POST) request, in EST-coaps the equivalent CoAP response code 2.05 (2.01) MUST be used. Response code HTTP 202 in EST is mapped to CoAP _.__. In [I-D.hartke-core-pending] it is specified how multiple concurrently

open requests may be handled. All other HTTP 2xx response codes are not used by EST. For the following HTTP 4xx error codes that may occur: 400, 401, 403, 404, 405, 406, 412, 413, 415; the equivalent CoAP response code for EST-coaps is 4.xx. For the HTTP 5xx error codes: 500, 501, 502, 503, 504 the equivalent CoAP response code is 5.xx.

3.4. Message fragmentation

DTLS defines fragmentation only for the handshake part and not for secure data exchange (DTLS records). [RFC6347] states that to avoid using IP fragmentation, which involves error-prone datagram reconstitution, invokers of the DTLS record layer SHOULD size DTLS records so that they fit within any Path MTU estimates obtained from the record layer. In addition, invokers residing on a 6LoWPAN over IEEE 802.15.4 network SHOULD attempt to size CoAP messages such that each DTLS record will fit within one or two IEEE 802.15.4 frames.

That is not always possible. Even though ECC certificates are small in size, they can vary greatly based on signature algorithms, key sizes, and OID fields used. For 256-bit curves, common ECDSA cert sizes are 500-1000 bytes which could fluctuate further based on the algorithms, OIDs, SANs and cert fields. For 384-bit curves, ECDSA certs increase in size and can sometimes reach 1.5KB. Additionally, there are times when the EST cacerts response from the server can include multiple certs that amount to large payloads. Section 4.6 of CoAP [RFC7252] describes the possible payload sizes: "if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size". Section 4.6 of [RFC7252] also suggests that IPv4 implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes. From [RFC0791] follows that the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Thus, even with ECC certs, EST-coaps messages can still exceed sizes in MTU of 1280 for IPv6 or 60-80 bytes for 6LoWPAN [RFC4919] as explained in section 2 of [RFC7959]. EST-coaps needs to be able to fragment EST messages into multiple DTLS datagrams. Fine-grained fragmentation of EST messages is essential.

To perform fragmentation in CoAP, [RFC7959] specifies the "Block1" option for fragmentation of the request payload and the "Block2" option for fragmentation of the return payload of a CoAP flow.

The BLOCK draft defines SZX in the Block1 and Block2 option fields. These are used to convey the size of the blocks in the requests or responses.

The CoAP client MAY specify the Block1 size and MAY also specify the Block2 size. The CoAP server MAY specify the Block2 size, but not the Block1 size. As explained in Section 1 of [RFC7959]), blockwise transfers SHOULD be used in Confirmable CoAP messages to avoid the exacerbation of lost blocks.

The Size1 response MAY be parsed by the client as a size indication of the Block2 resource in the server response or by the server as a request for a size estimate by the client. Similarly, Size2 option defined in BLOCK should be parsed by the server as an indication of the size of the resource carried in Block1 options and by the client as a maximum size expected in the 4.13 (Request Entity Too Large) response to a request.

Examples of fragmented messages are shown in Appendix C.

3.5. Deployment limits

Although EST-coaps paves the way for the utilization of EST for constrained devices on constrained networks, some devices will not have enough resources to handle the large payloads that come with EST-coaps. The specification of EST-coaps is intended to ensure that EST works for networks of constrained devices that choose to limit their communications stack to UDP/CoAP. It is up to the network designer to decide which devices execute the EST protocol and which not.

4. Discovery and URI

EST-coaps is targeted to low-resource networks with small packets. Saving header space is important and an additional EST-coaps URI is specified that is shorter than the EST URI.

In the context of CoAP, 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 "ace.est" [RFC6690]. Upon success, the return payload will contain the root resource of the EST resources. It is up to the implementation to choose its root resource; throughout this document the example root resource /est is used. The example below shows the discovery of the presence and location of management data.

```
  REQ: GET /.well-known/core?rt=ace.est

  RES: 2.05 Content
</est>; rt="ace.est"
```

The additional EST-coaps server URIs differ from the EST URI by replacing the scheme https by coaps and by specifying a shorter resource path names:

coaps://www.example.com/est/short-name

The CoAP short URI exists next to the URI defined in [RFC7030].

coaps://www.example.com/.well-known/est/est-name
OR
coaps://www.example.com/.well-known/est/ArbitraryLabel/est-name

Figure 5 in section 3.2.2 of [RFC7030] enumerates the operations and corresponding paths which are supported by EST. Table 1 provides the mapping from the EST URI path to the shorter EST-coaps URI path.

EST	EST-coaps
/cacerts	/crts
/simpleenroll	/sen
/simplereenroll	/sren
/csrattrs	/att
/serverkeygen	/skg

Table 1

When discovering the root path for the EST resources, the server MAY return the full resource paths and the used content types. This is useful when multiple content types are specified for EST-coaps server. For example, the following more complete response is possible.

REQ: GET /.well-known/core?rt=ace.est

RES: 2.05 Content
</est>; rt="ace.est"
</est/crts>; rt="ace.est";ct=TBD1
</est/sen>; rt="ace.est";ct=TBD1 TBD4
</est/sren>; rt="ace.est";ct=TBD1 TBD4
</est/att>; rt="ace.est";ct=TBD4
</est/skg>; rt="ace.est";ct=TBD1 TBD4 TBD2

The return of the content-types allows the client to choose the most appropriate one from multiple content types.

5. DTLS Transport Protocol

EST-coaps depends on a secure transport mechanism over UDP that can secure (confidentiality, authenticity) the CoAP messages exchanged.

DTLS is one such secure protocol. When "TLS" is referred to in the context of EST, it is understood that in EST-coaps, security is provided using DTLS instead. No other changes are necessary (all provisional modes etc. are the same as for TLS).

CoAP was designed to avoid fragmentation. DTLS is used to secure CoAP messages. However, fragmentation is still possible at the DTLS layer during the DTLS handshake when using ECC ciphersuites. If fragmentation is necessary, "DTLS provides a mechanism for fragmenting a handshake message over a number of records, each of which can be transmitted separately, thus avoiding IP fragmentation" [RFC6347].

CoAP and DTLS can provide proof of identity for EST-coaps clients and server with simple PKI messages conformant to section 3.1 of [RFC5272]. EST-coaps supports the certificate types and Trust Anchors (TA) that are specified for EST in section 3 of [RFC7030].

Channel-binding information for linking proof-of-identity with connection-based proof-of-possession is optional for EST-coaps. When proof-of-possession is desired, a set of actions are required regarding the use of tls-unique, described in section 3.5 in [RFC7030]. The tls-unique information translates to the contents of the first "Finished" message in the TLS handshake between server and client [RFC5929]. The client is then supposed to add this "Finished" message as a ChallengePassword in the attributes section of the PKCS#10 Request Info to prove that the client is indeed in control of the private key at the time of the TLS session when performing a /simpleenroll, for example. In the case of EST-coaps, the same operations can be performed during the DTLS handshake. In the event of handshake message fragmentation, the Hash of the handshake messages used in the MAC calculation of the Finished message

```
PRF(master_secret, finished_label, Hash(handshake_messages))
   [0..verify_data_length-1];
```

MUST be computed as if each handshake message had been sent as a single fragment [RFC6347].

In a constrained CoAP environment, endpoints can’t afford to establish a DTLS connection for every EST transaction. Authenticating and negotiating DTLS keys requires resources on low-end endpoints and consumes valuable bandwidth. The DTLS connection

SHOULD remain open for persistent EST connections. For example, an EST cacerts request that is followed by a simpleenroll request can use the same authenticated DTLS connection. Given that after a successful enrollment, it is more likely that a new EST transaction will take place after a significant amount of time, the DTLS connections SHOULD only be kept alive for EST messages that are relatively close to each other.

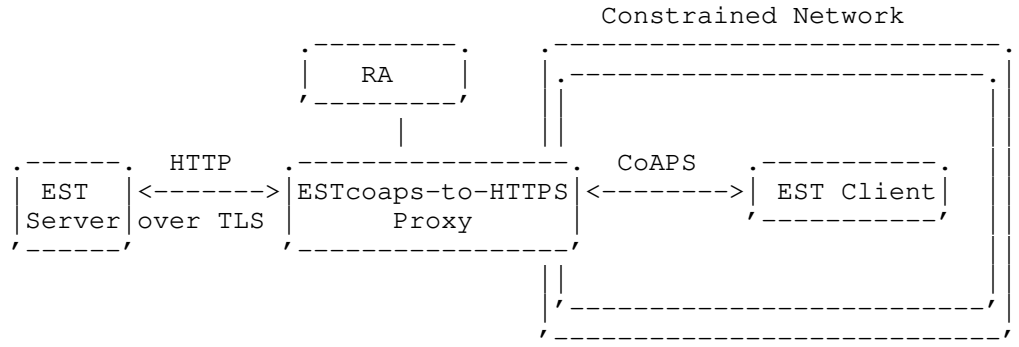
Support for Observe CoAP options [RFC7641] is out-of-scope for this document. Observe options could be used by the server to notify clients about a change in the cacerts or csr attributes (resources) and might be an area of future work.

6. Proxying

In real-world deployments, the EST server will not always reside within the CoAP boundary. The EST-server can exist outside the constrained network in a non-constrained network that supports TLS/HTTP. In such environments EST-coaps is used by the client within the CoAP boundary and TLS is used to transport the EST messages outside the CoAP boundary. A proxy entity at the edge is required to operate between the CoAP environment and the external HTTP network. The ESTcoaps-to-HTTPS proxy SHOULD terminate EST-coaps downstream and initiate EST connections over TLS upstream.

One possible use-case, shown in one figure below, is expected to be deployed in practice:

- A proxy between any EST-client and EST-server



ESTcoaps-to-HTTPS proxy at the CoAP boundary.

Table 1 contains the URI mapping between the EST-coaps and EST the proxy SHOULD adhere to. Section 7 of [RFC8075] and Section 3.3 define the mapping between EST-coaps and HTTP response codes, that

determines how a proxy translates CoAP response codes from/to HTTP status codes. The mapping from Content-Type to media type is defined in Section 8. The conversion from binary to BSD64 needs to be done in the proxy. Conversion is possible because a TLS link exists between EST-coaps-to-HTTP proxy and EST server and a corresponding DTLS linked exists between EST-coaps-to-HTTP proxy and EST client.

Due to fragmentation of large messages into blocks, an EST-coaps-to-HTTP proxy SHOULD reassemble the BLOCKs before translating the binary content to BSD64, and consecutively relay the message upstream into the HTTP environment.

For the discovery of the EST server by the EST client in the coap environment, the EST-coaps-to-HTTP proxy MUST announce itself according to the rules of Section 4. The available functions of the proxies MUST be announced with as many resource paths. The discovery of EST server in the http environment follow the rules specified in [RFC7030].

[EDNOTE: PoP will be addressed here.]

A proxy SHOULD authenticate the client downstream and it should be authenticated by the EST server or CA upstream. The Registration Authority (RA) is necessary to (re-)create the secure connection from DTLS to TLS and vice versa. A trust relationship needs to be pre-established between the proxy and the EST servers to be able to proxy these connections on behalf of various clients.

[EDNOTE: To add more details about trust relations in this section.]

7. Parameters

[EDNOTE: This section to be populated. It will address transmission parameters described in sections 4.7 and 4.8 of the CoAP draft. EST does not impose any unique parameters that affect the CoAP parameters in Table 2 and 3 in the CoAP draft but the ones in CoAP could be affecting EST. For example, the processing delay of CAs could be less then 2s, but in this case they should send a CoAP ACK every 2s while processing.]

8. IANA Considerations

8.1. Content-Format registry

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are needed for the below media types. These can be registered either in the Expert Review range (0-255) or IETF Review range (256-9999).

1.

   * application/pkcs7-mime
   * Type name: application
   * Subtype name: pkcs7-mime
   * ID: TBD1
   * Required parameters: None
   * Optional parameters: None
   * Encoding considerations: binary
   * Security considerations: As defined in this specification
   * Published specification: [RFC5751]
   * Applications that use this media type: EST

2.

   * application/pkcs8
   * Type name: application
   * Subtype name: pkcs8
   * ID: TBD2
   * Required parameters: None
   * Optional parameters: None
   * Encoding considerations: binary
   * Security considerations: As defined in this specification
   * Published specification: [RFC5958]
   * Applications that use this media type: EST

3.

   * application/csrattrs

* Type name: application
* Subtype name: csrattrs
* ID: TBD3
* Required parameters: None
* Optional parameters: None
* Encoding considerations: binary
* Security considerations: As defined in this specification
* Published specification: [RFC7030]
* Applications that use this media type: EST

4.

* application/pkcs10
* Type name: application
* Subtype name: pkcs10
* ID: TBD4
* Required parameters: None
* Optional parameters: None
* Encoding considerations: binary
* Security considerations: As defined in this specification
* Published specification: [RFC5967]
* Applications that use this media type: EST

8.2. Resource Type registry

Additions to the sub-registry "CoAP Resource Type", within the "CoRE Parameters" registry are needed for a new resource type.

o rt="ace.est" needs registration with IANA.

9. Security Considerations

9.1. proxy considerations

The proxy proposed in Section 6 must be deployed with great care, and only when the recommended connections are impossible.

[EDNOTE: To add more details about trust relations through proxies in this section.]

9.2. EST server considerations

The security considerations of section 6 of [RFC7030] are only partially valid for the purposes of this document. As HTTP Basic Authentication is not supported, the considerations expressed for using passwords do not apply.

Given that the client has only limited resources and may not be able to generate sufficiently random keys to encrypt its identity, it is possible that the client uses server generated private/public keys to encrypt its certificate. The transport of these keys is inherently risky. A full probability analysis MUST be done to establish whether server side key generation enhances or decreases the probability of identity stealing.

When a client uses the Implicit TA database for certificate validation, the client cannot verify that the implicit data base can act as an RA. It is RECOMMENDED that such clients include "Linking Identity and POP Information" Section 5 in requests (to prevent such requests from being forwarded to a real EST server by a man in the middle). It is RECOMMENDED that the Implicit Trust Anchor database used for EST server authentication be carefully managed to reduce the chance of a third-party CA with poor certification practices from being trusted. Disabling the Implicit Trust Anchor database after successfully receiving the Distribution of CA certificates response (Section 4.1.3 of [RFC7030]) limits any vulnerability to the first DTLS exchange.

In accordance with [RFC7030], TLS cipher suites that include "_EXPORT_" and "_DES_" in their names MUST NOT be used. More information about recommendations of TLS and DTLS are included in [RFC7525].

As described in CMC, Section 6.7 of [RFC5272], "For keys that can be used as signature keys, signing the certification request with the private key serves as a POP on that key pair". The inclusion of tls-unique in the certification request links the proof-of-possession to

the TLS proof-of-identity. This implies but does not prove that the authenticated client currently has access to the private key.

Regarding the CSR attributes that the CA may list for inclusion in an enrollment request, an adversary could exclude attributes that a server may want, include attributes that a server may not want, and render meaningless other attributes that a server may want. The CA is expected to be able to enforce policies to recover from improper CSR requests.

Interpreters of ASN.1 structures should be aware of the use of invalid ASN.1 length fields and should take appropriate measures to guard against buffer overflows, stack overruns in particular, and malicious content in general.

10. Acknowledgements

The authors are very grateful to Klaus Hartke for his detailed explanations on the use of Block with DTLS. The authors would like to thank Esko Dijk and Michael Verschoor for the valuable discussions that helped in shaping the solution. They would also like to thank Peter Panburana from Cisco for his feedback on technical details of the solution. Constructive comments were received from Eliot Lear, Jim Schaad, Hannes Tschofenig, and Julien Vermillard.

11. Change Log

-00:

copied from vanderstok-ace-coap-est-04

12. References

12.1. Normative References

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

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <https://www.rfc-editor.org/info/rfc5272>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <https://www.rfc-editor.org/info/rfc5751>.

[RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <https://www.rfc-editor.org/info/rfc5967>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

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

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

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

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

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

12.2. Informative References

[I-D.hartke-core-pending]
Stok, P. and K. Hartke, ""Pending" Responses for the Constrained Application Protocol (CoAP)", draft-hartke-core-pending-02 (work in progress), February 2018.

[I-D.ietf-6tisch-dtsecurity-secure-join]
Richardson, M., "6tisch Secure Join protocol", draft-ietf-6tisch-dtsecurity-secure-join-01 (work in progress), February 2017.

[I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-04 (work in progress), October 2017.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-11 (work in progress), February 2018.

[ieee802.15.4]
Institute of Electrical and Electronics Engineers, "IEEE Standard 802.15.4-2006", 2006.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <https://www.rfc-editor.org/info/rfc4492>.

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

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

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

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <https://www.rfc-editor.org/info/rfc5929>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <https://www.rfc-editor.org/info/rfc5958>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <https://www.rfc-editor.org/info/rfc6090>.

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

[RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS", RFC 7251, DOI 10.17487/RFC7251, June 2014, <https://www.rfc-editor.org/info/rfc7251>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

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

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

Appendix A. EST messages to EST-coaps

This section takes all examples from Appendix A of [RFC7030], changes the payload from Base64 to binary and replaces the http headers by their CoAP equivalents.

The corresponding CoAP headers are only shown in Appendix A.1. Creating CoAP headers are assumed to be generally known.

[EDNOTE: The payloads of the examples need to be re-generated with appropriate tools and example certificates.]

A.1. cacerts

In EST-coaps, a coaps cacerts IPv4 message can be:

GET coaps://[192.0.2.1:8085]/est/crts

The corresponding CoAP header fields are shown below. The use of block and DTLS are worked out in Appendix C.

```
  Ver = 1
  T = 0 (CON)
  Code = 0x01 (0.01 is GET)
  Options
   Option1 (Uri-Host)
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Uri-Port)
     Option Delta = 0x4  (option nr = 4+3=7)
     Option Length = 0x4
     Option Value = 8085
   Option3 (Uri-Path)
     Option Delta = 0x4   (option nr = 7+4= 11)
     Option Length = 0x9
     Option Value = /est/crts
  Payload = [Empty]
```

A 2.05 Content response with a cert in EST-coaps will then be:

2.05 Content (Content-Format: application/pkcs7-mime)
{payload}

with CoAP fields

```
  Ver = 1
  T = 2 (ACK)
  Code = 0x45 (2.05 Content)
  Options
    Option1 (Content-Format)
      Option Delta = 0xC  (option nr = 12)
      Option Length = 0x2
      Option Value = TBD1 (defined in this document)

  Payload =
30233906092a6206734107028c2a3023260201013100300b06092a6206734107018
c0c3020bb302063c20102020900a61e75193b7acc0d06092a620673410105050030
1b3119301706035504031310657374578616d706c654341204f774f301e170d313
330353039303353333315a170d313430353039303353333315a301b3119301706
```

```
03550403131065737445786 16d706c654341204f774f302062300d06092a6206734
10101050003204f0030204a022041003a923a2968bae4aae136ca4e2512c5200680
358482ac39d6f640e4574e654ea35f48b1e054c5da3372872f7a1e429f4edf39584
32efb2106591d3eb783c1034709f251fc86566bda2d541c792389eac4ec9e181f4b
9f596e5ef2679cc321542b11337f90a44df3c85f1516561fa968a1914f265bc0b82
76ebe3106a790d97d34c8c37c74fe1c30b396424664ac426284a9f6022e02693843
6880adfcd95c98ca1dfc2e6d75319b85d0458de28a9d13fb16d620fff7541f6a25d
7daf004355020301000130b040300f0603551d130101f10530030101fc1d0603551
d0e04160414084d321ca0135e77217a486b686b334b00e0603551d0f0101f104030
20106300d06092a62067341010505000320410023703b965746a0c2c978666d787a
94f89b495a11f0d369b28936ec2475c0f0855c8e83f823f2b871a1d92282f323c45
904ba008579216cf5223b8b1bc425a0677262047f7700240631c17f3035d1c3780b
2385241cba1f4a6e98e6be6820306b3a786de5a557795d1893822347b5f825d34a7
ad2876f8feba4d525b31066f6505796f71530003431a3e6bbfe788b4565029a7e20
a51107677552586152d051e8eebf383e92288983421d5c5652a4870c3af74b9bdbe
d6b462e2263d30f6d3020c330206bc20102020101300d06092a6206734101050500
301b311930170603550403131065737445786 16d706c654341204f774f301e170d3
13330353039303353333325a170d313430353039303353333325a301b31193017
060355040313106573744578616d706c654341204e774f302062300d06092a62067
34101010500032 04f0030204a02204100ef6b677a3247c1fc03d2b9baf113e5e7e1
1f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a7229283a790
8751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b7bd94338
d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562c4f5abb7
b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c768d03b8
076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c14476c37de0f
55033f192a5ad21f9a2a71c20301000134b050300e0603551d0f0101f104030204c
1d0603551d0e04160414112966e304761732fbfe6a2c823c301f0603551d2304183
0165084d321ca0135e77217a486b686b334b00d06092a62067341010500032041
00b382ba3355a50e287bae15758b3beff63d34d3e357b90031495d018868e49589b
9faf46a4ad49b1d35b06ef380106677440934663c2cc111c183655f4dc41c0b3401
123d35387389db91f1e1b4131b16c291d35730b3f9b33c7475124851555fe5fc647
e8fd029605367c7e01281bf6617110021b0d10847dce0e9f0ca6c764b6334784055
172c3983d1e3a3a82301a54fcc9b0670c543a1c747164619101ff23b240b2a26394
c1f7d38d0e2f4747928ece5c34627a075a8b3122011e9d9158055c28f020c330206
bc20102020102300d06092a62067341010505003 01b311930170603550403131065
73744578616d706c654341204e774e301e170d313330353039303353333325a170
d313430353039303353333325a301b311930170603550403131065737445786 16d
706c654341204f774e302062300d06092a62067341010105000 3204f0030204a022
041003a923a2968bae4aae136ca4e2512c5200680358482ac39d6f640e4574e654e
a35f48b1e054c5da3372872f7a1e429f4edf3958432efb2106591d3eb783c103470
9f251fc86566bda2d541c792389eac4ec9e181f4b9f596e5ef2679cc321542b1133
7f90a44df3c85f1516561fa968a1914f265bc0b8276ebe3106a790d97d34c8c37c7
4fe1c30b396424664ac426284a9f6022e026938436880adfcd95c98ca1dfc2e6d75
319b85d0458de28a9d13fb16d620fff7541f6a25d7daf004355020301000134b050
300e0603551d0f0101f104030204c1d0603551d0e04160414084d321ca0135e7721
7a486b686b334b01f0603551d230418301653112966e304761732fbfe6a2c823c30
0d06092a62067341010505000320 41002e106933a443070acf5594a3a584d08af7e
06c295059370a06639eff9bd418d13bc25a298223164a6cf1856b11a81617282e4a
```

```
410d82ef086839c6e23569032276306545535 1e4c596acc7c016b225dec094706c2
a10608f403b10821984c7c152343b18a768c2ad30238dc45dd653ee6092b0d5cd4c
2f7d236043269357f76d13f95fb5f00d0e19263c6833948e1ba612ce8197af650e2
5d882c12f4b6b9b67252c608ef064aca3f9bc867d71172349d510bb7651cd438837
73d927deb41c4673020bb302063c201020209009b9dda3324700d06092a62067341
01050500301b311930170603550403131065737445786 16d706c654341204e774e3
01e170d313330353039303353333325a170d313430353039303353333325a301b
311930170603550403131065737445786 16d706c654341204e774e302062300d060
92a62067341010105000 3204f0030204a02204100ef6b677a3247c1fc03d2b9baf1
13e5e7e11f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a722
9283a7908751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b
7bd94338d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562
c4f5abb7b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c
768d03b8076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c1447
6c37de0f55033f192a5ad21f9a2a71c20301000130b040300f0603551d130101f10
530030101fc1d0603551d0e04160414112966e304761732fbfe6a2c823c300e0603
551d0f0101f10403020106300d06092a62067341010505000 3204100423f06d4b76
0f4b42744a279035571696f272a0060f1325a40898509601ad14004f652db6312a1
475c4d7cd50f4b269035585d7856c5337765a66b38462d5bdaa7778aab24bbe2815
e37722cd10e7166c50e75ab75a1271324460211991e7445a2960f47351a1a629253
34119794b90e320bc730d6c1bee496e7ac125ce9a1eca595a3a4c54a865e6b623c9
247bfd0a7c19b56077392555c955e233642bec643ae37c166c5e221d797aea3748f
0391c8d692a5cf9bb71f6d0e37984d6fa673a30d0c006343116f58403100
```

A.2. csrattrs

In the following valid /csrattrs exchange, the EST-coaps client authenticates itself with a certificate issued by the connected CA.

The initial DTLS handshake is identical to the enrollment example. The IPv6 CoAP GET request looks like:

```
REQ:
GET coaps://[2001:db8::2:1]:61616/est/att
```

A 2.05 Content response contains attributes which are relevant for the authenticated client. In this example, the EST-coaps server two attributes that the client can ignore when they are unknown to him.:

A.3. enroll / reenroll

[EDNOTE: We might need a new Option for the Retry-After response message. We might need a new Option for the WWW-Authenticate response.]

During the Enroll/Reenroll exchange, the EST-coaps client uses a CSR (PKCS#10) request in the POST request payload.

After verification of the certificate by the server, a 2.05 Content response with the issued certificate will be returned.

POST [2001:db8::2:1]:61616/est/sen
(Content-Format: application/pkcs10)
30208530206d020100301f311d301b06035504031314646566d6f7374657034203
133363831343133353230206230 0d06092a620673410101050003204f0030204a
022041005d9f4dffd3c5949f646a9584367778560950b355c35b8e34726dd3764
54231734795b4c09b9c6d75d408311307a81f7adef7f5d241f7d5be85620c5d44
38bbb4242cf215c167f2ccf36c364ea2618a62f0536576369d6304e6a96877224
7d86824f079faac7a6f694cfda5b84c42087dc062d462190c525813f210a036a7
37b4f30d8891f4b75559fb72752453146332d51c937557716ccec624f5125c3a4
447ad3115020048113fef54ad554ee88af09a2583aac9024075113db4990b1786
b871691e0f020301000187 01f06092a620673410907311213102b72724369722f
372b45597535305434300d06092a620673410105050003204100441b40177a3a6
5501487735a8ad5d3827a4eaa867013920e2afcda87aa81733c7c0353be47e1bf
a7cda5176e7ccc6be22ae03498588d5f2de3b143f2b1a6175ec544e8e7625af6b
836fd4416894c2e55ea99c6606f69075d6d53475d410729aa6d806afbb9986caf
7b844b5b3e4545f19071865ada007060cad6db26a592d4a7bda7d586b68110962
17071103407553155cddc75481e272b5ed553a8593fb7e25100a6f7605085dab4
fc7e0731f0e7fe305703791362d5157e92e6b5c2e3edbcadb40

RET:
2.05 Content (Content-Format: application/pkcs7-mime)
3020f806092a620673410702832930 20e50201013100300b06092a62067341070
1830b3020c730206fc20102020115300d06092a620673410105050030 1b311930
17060355040313106573744578616d706c654341204e774e301e170d313330353
0393233313535335a170d313430353039323331353535a301f311d301b060355
0403131464656d6f737465703420313336383134313335323020623 00d06092a6
206734101010500032 04f0030204a022041005d9f4dffd3c5949f646a95843677
78560950b355c35b8e34726dd376454231734795b4c09b9c6d75d408311307a81
f7adef7f5d241f7d5be85620c5d4438bbb4242cf215c167f2ccf36c364ea2618a
62f0536576369d6304e6a968772247d86824f079faac7a6f694cfda5b84c42087
dc062d462190c525813f210a036a737b4f30d8891f4b75559fb72752453146332
d51c937557716ccec624f5125c3a4447ad3115020048113fef54ad554ee88af09
a2583aac9024075113db4990b1786b871691e0f020301000134b050300e060355
1d0f0101f104030204c1d0603551d0e04160414e81d0788aa2710304c5ecd4d1e
065701f0603551d230418301653112966e304761732fbfe6a2c823c300d06092a
620673410105050003204100291 0d86f2ffeeb914c046816871de601567d291b4
3fabee0f0e8ff81cea27302a7133e20e9d04029866a8963c7d14e26fbe8a0ab1b
77fbb1214bbcdc906fbc381137ec1de685f79406c3e416b8d82f97174bc691637
5a4e1c4bf744c7572b4b2c6bade9fb35da786392ee0d95e3970542565f3886ad6
7746d1b12484bb02616e63302dc371dc6006e431fb7c457598dd204b367b0b3d3
258760a303f1102db26327f929b7c5a60173e1799491b69150248756026b80553
171e4733ad3d13c0103100

[EDNOTE: If POP is used, make sure tls-unique in the CSR is a valid HMAC output.]

A.4. serverkeygen

During this valid /serverkeygen exchange, the EST-coaps client authenticates itself using the certificate provided by the connected CA.

[EDNOTE: the client incudes a CSR with a public key that the server should ignore, so we need a content-format here.]

[EDNote: If POP is used, make sure tls-unique in the CSR is a valid HMAC output.]

The initial DTLS handshake is identical to the enrollment example. The CoAP GET request looks like:

POST coaps://[192.0.2.1:8085]/est/skg
302081302069020100305b313e303c06035504031335736572766572 4b6579476
56e207265712062792063 6c69656e7420696e2064656d6f207374657020313220
313336383134313935353119301706035504051310504944 3a576964676574205
34e3a3130302062300d06092a620673410101050003204f0030204a02204100f4
dfa6c03f7f2766b23776c333d2c0f9d1a7a6ee36d01499bbe6f075d1e38a57e98
ecc197f51b75228454b7f19652332de5e52e4a974c6ae34e1df80b33f15f47d3b
cbf76116bb0e4d3e04a9651218a476a13fc186c2a255e4065ff7c271cff104e47
31fad53c22b21a1e5138bf9ad0187314ac39445949a48805392390e78c7659621
6d3e61327a534f5ea7721d2b1343c7362b37da502717cfc2475653c7a3860c5f4
0612a5db6d33794d755264b6327e3a3263b149628585b85e57e42f6b3277591b0
2030100018701f06092a620673410907311213106446734158 6d4a6e6a6f6b427
4447672300d06092a62067341010505000320410047 2d11007e5a2b2c2023d47a
6d71d046c307701d8ebc9e47272713378390b4ee321462a3dbe54579f5a514f6f
4050af497f428189b63655d03a194ef729f101743e5d03fbc6ae1e84486d1300a
f9288724381909188c851fa9a5059802eb64449f2a3c9e441353d136768da27ff
4f277651d676a6a7e51931b08f56135a2230891fd184960e1313e7a1a9139ed19
28196867079a456cd2266cb754a45151b7b1b939e381be333fea61580fe5d25bf
4823dbd2d6a98445b46305c10637e202856611

RET:
2.05 Content (Content-Format: application/pkcs8)
30213e020100300d06092a620673410101050004212830212402010002204 1003
c0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274
dd01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a1
1bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c
0c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d5
45e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704
efec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e31083010
001022041004e6b3f78b7791d6377f33117c17844531c81111fb8000282816264
915565bc7c3f3f643b537a2c69140a31c22550fa97e5132c61b74166b68626704
260620333050f510096b6570f5880e7e1c15dc0ca6ce2b5f187e2325da14ab705
ad004717f3b2f779127b5c535e0cee6a343b502722f2397a26126e0af606b5aa7

```
f96313511c0b7eb26354f91b82269de62757e3def807a6afdf83ddcbb0614bb7c
542e6975d6456554e7bd9988fbd1930cd44d0e01ee9182ca54539418653150254
1ad1a2a11e5021040bfce554b642c29131e7d65455e83c5406d76771912f758f5
ee3ee36af386f38ffa313c0f661880c5a2b0970485d36f528e7f77a2e55b4ad76
1242d1c2f75939c8061217d31491d305d3e07d6161c43e26f7de4477b1811de92
33dc75b426302104015bf48ac376f52887813461fc54635517bcb67293837053e
8ce1a33da7a35565a75a370dc14555b5316cb55742380350774d769d151ff0456
0214389a232a2258326163167504cfce44cd316f63bb8a52da53a4cb74fd87194
c0844881f791f23b0813ea0921325edd14459d41c8a1593f04316388e40b35fef
7d2a195a5930fa54774427ac821eee2c62790d2c17bd192af794c611011506557
83d4efe22185cbd83368786f2b1e68a5a27067e321066f0217b4b6d7971a3c21a
241366b7907187583b511102103369047e5cce0b65012200df5ec697b5827575c
db6821ff299d6a69574b31ddf0fbe9245ea2f74396c24b3a7565067e41366423b
5bdd2b2a78194094dbe333f493d159b8e07722f2280d48388db7f1c9f0633bb0e
173de2c3aa1f200af535411c7090210401421e2ea217e37312dcc606f453a6634
f3df4dc31a9e910614406412e70eec9247f10672a500947a64356c015a845a7d1
50e2e3911a2b3b61070a73247166da10bb45474cc97d1ec2bc392524307f35118
f917438f607f18181684376e13a39e07
--estServerExampleBoundary
3020c506092a620673410702833630 20f20201013100300b06092a62067341070
183183020d430207cc20102020116300d06092a620673410105050030 1b311930
17060355040313106573744578616d706c654341204e774e301e170d313330353
0393233323535365a170d313430353039323323535365a302c312a3028060355
040313217365727665727369646520 6b65792067656e6572617465642072657 37
06f6e7365302062300d06092a620673410101050003204f0030204a022041003c
0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274d
d01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a11
bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c0
c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d54
5e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704e
fec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e310830100
0134b050300e0603551d0f0101f104030204c1d0603551d0e04160414764b1bd5
e69935626e476b195a1a8c1f0603551d230418301653112966e304761732fbfe6
a2c823c300d06092a620673410105050003204100474e5100a9cdaaa813b30f48
40340fb17e7d6d6063064a5a7f2162301c464b5a8176623dfb1a4a484e618de1c
3c3c5927cf590f4541233ff3c251e772a9a3f2c5fc6e5ef2fe155e5e385deb846
b36eb4c3c7ef713f2d137ae8be4c022715fd033a818d55250f4e6077718180755
a4fa677130da60818175ca4ab2af1d15563624c51e13dfdcf381881b72327e2f4
9b7467e631a27b5b5c7d542bd2edaf78c0ac294f3972278996bdf673a334ff74c
84aa7d65726310252f6a4f41281ec10ca2243864e3c5743103100
```

Without the DecryptKeyIdentifier attribute, the response has no additional encryption beyond DTLS. [EDNOTE: Add comment about deriving symmetric keys by using the TLS KEM draft.]

The response contains first a preamble that can be ignored. The EST-coaps server can use the preamble to include additional explanations, like ownership or support information

Appendix B. Encoding for server side key generation

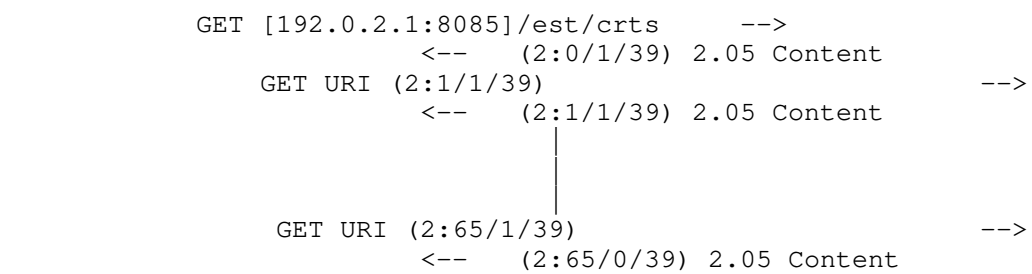
Sever side key generation for CoAP can be implemented efficiently using multipart encoding

[EDNOTE: text to be written.]

Appendix C. EST-coaps Block message examples

This section provides a detailed example of the messages using DTLS and BLOCK option Block2. The minimum PMTU is 1280 bytes, which is the example value assumed for the DTLS datagram size. The example block length is taken as 64 which gives an SZX value of 2.

The following is an example of a valid /cacerts exchange over DTLS. The content length of the cacerts response in appendix A.1 of [RFC7030] is 4246 bytes using base64. This leads to a length of 2509 bytes in binary. The CoAP message adds around 10 bytes, the DTLS record 29 bytes. To avoid IP fragmentation, the CoAP block option is used and an MTU of 127 is assumed to stay within one IEEE 802.15.4 packet. To stay below the MTU of 127, the payload is split in 39 packets with a payload of 64 bytes each, followed by a packet of 13 bytes. The client sends an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP Request 40 times. The server returns an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP response. The CoAP request-response exchange with block option is shown below. Block option is shown in a decomposed way indicating the kind of Block option (2 in this case because used in the response) followed by a colon, and then the block number (NUM), the more bit (M = 0 means last block), and block size exponent (2**(SZX+4)) separated by slashes. The Length 64 is used with SZX= 2 to avoid IP fragmentation. The CoAP Request is sent with confirmable (CON) option and the content format of the Response is /application/ cacerts.



For further detailing the CoAP headers of the first two blocks are written out.

The header of the first GET looks like:

```
  Ver = 1
  T = 0 (CON)
  Code = 0x01 (0.1 GET)
  Options
   Option1 (Uri-Host)
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Uri-Port)
     Option Delta = 0x4   (option nr = 3+4=7)
     Option Length = 0x4
     Option Value = 8085
   Option3 (Uri-Path)
     Option Delta = 0x4    (option nr = 7+4=11)
     Option Length = 0x9
     Option Value = /est/crts
  Payload = [Empty]
```

The header of the first response looks like:

```
  Ver = 1
  T = 2 (ACK)
  Code = 0x45 (2.05 Content.)
  Options
    Option1 (Content-Format)
      Option Delta = 0xC  (option 12)
      Option Length = 0x2
      Option Value = TBD1
    Option2 (Block2)
      Option Delta = 0xB  (option 23 = 12 + 11)
      Option Length = 0x1
      Option Value = 0x0A (block number = 0, M=1, SZX=2)
  Payload =
30233906092a6206734107028c2a3023260201013100300b06092a6206734107018
c0c3020bb302063c20102020900a61e75193b7acc0d06092a6206734101
```

The second Block2:

```
     Ver = 1
     T = 2 (means ACK)
     Code = 0x45 (2.05 Content.)
     Options
       Option1 (Content-Format)
         Option Delta = 0xC   (option 12)
         Option Length = 0x2
         Option Value = TBD1
       Option2 (Block2)
         Option Delta = 0xB  (option 23 = 12 + 11)
         Option Length = 0x1
         Option Value = 0x1A (block number = 1, M=1, SZX=2)
     Payload =
   05050030
   1b311930170603550403131065737445786l6d706c654341204f774f301e170d313
   330353039303353333315a170d313430353039303353333315a
```

The 40th and final Block2:

```
     Ver = 1
     T = 2 (means ACK)
     Code = 0x21
     Options
       Option1 (Content-Format)
         Option Delta = 0xC   (option 12)
         Option Length = 0x2
         Option Value = TBD1
       Option2 (Block2)
         Option Delta = 0xB  (option 23 = 12 + 11)
         Option Length = 0x2
         Option Value = 0x272 (block number = 39, M=0, SZX=2)
     Payload = 73a30d0c006343116f58403100
```

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Sandeep S. Kumar
Philips Lighting Research
High Tech Campus 7
Eindhoven 5656 AE
NL

Email: ietf@sandeep.de

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
URI: http://www.sandelman.ca/

Martin Furuhed
Nexus Group

Email: martin.furuhed@nexusgroup.com

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista, Stockholm 16440
SE

Email: shahid@sics.se

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson

March 05, 2018

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-ietf-core-coap-pubsub-04

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 6, 2018.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Architecture 4
3.1. CoAP Pub/sub Architecture 4
3.2. CoAP Pub/sub Broker 4
3.3. CoAP Pub/sub Client 5
3.4. CoAP Pub/sub Topic 5
3.5. Brokerless Pub/sub 5
4. CoAP Pub/sub REST API 6
4.1. DISCOVERY 6
4.2. CREATE 8
4.3. PUBLISH 10
4.4. SUBSCRIBE 13
4.5. UNSUBSCRIBE 14
4.6. READ 16
4.7. REMOVE 17
5. CoAP Pub/sub Operation with Resource Directory 18
6. Sleep-Wake Operation 19
7. Simple Flow Control 19
8. Security Considerations 20
9. IANA Considerations 21
9.1. Resource Type value 'core.ps' 21
9.2. Resource Type value 'core.ps.discover' 21
9.3. Response Code value '2.07' 21
10. Acknowledgements 21
11. References 22
11.1. Normative References 22
11.2. Informative References 22
Authors' Addresses 23

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports machine-to-machine communication across networks of constrained devices. CoAP uses a request/response model where clients make requests to servers in order to request actions on resources. Depending on the situation the same device may act either as a server or a client.

One important class of constrained devices includes devices that are intended to run for years from a small battery, or by scavenging energy from their environment. These devices have limited

reachability because they spend most of their time in a sleeping state with no network connectivity. Devices may also have limited reachability due to certain middle-boxes, such as Network Address Translators (NATs) or firewalls. Such middle-boxes often prevent connecting to a device from the Internet unless the connection was initiated by the device.

This document specifies the means for nodes with limited reachability to communicate using simple extensions to CoAP. The extensions enable publish-subscribe communication using a broker node that enables store-and-forward messaging between two or more nodes. Furthermore the extensions facilitate many-to-many communication using CoAP.

2. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format [RFC6570] is used to describe the REST interfaces defined in this specification.

This specification makes use of the following additional terminology:

Publish-Subscribe (pub/sub): A messaging paradigm where messages are published to a broker and potential receivers can subscribe to the broker to receive messages. The publishers do not (need to) know where the message will be eventually sent: the publications and subscriptions are matched by a broker and publications are delivered by the broker to subscribed receivers.

CoAP pub/sub service: A group of REST resources, as defined in this document, which together implement the required features of this specification.

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 and pending notifications.

CoAP pub/sub Client: A CoAP client which is capable of publish or subscribe operations as defined in this specification.

Topic: A unique identifier for a particular item being published and/or subscribed to. A broker uses the topics to match subscriptions to publications. A topic is a valid CoAP URI as defined in [RFC7252]

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 REST API 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 of state update representations between certain CoAP pub/sub Clients. Clients Subscribe to topics upon which representations are Published by other Clients, which are forwarded by the Broker to the subscribing clients. A CoAP pub/sub Broker may be used as a REST resource proxy, retaining the last published representation to supply in response to Read requests from Clients.

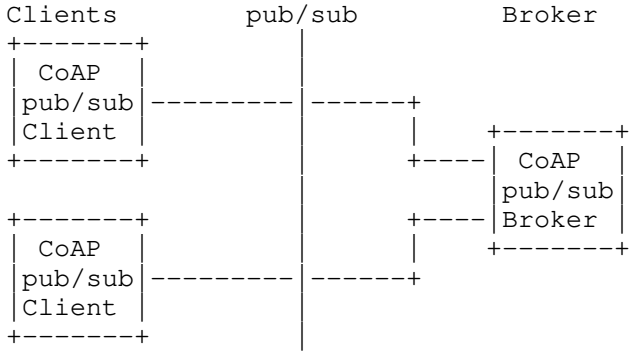


Figure 1: CoAP pub/sub Architecture

3.2. CoAP Pub/sub Broker

A CoAP pub/sub Broker is a CoAP Server that exposes a REST API for clients to use to initiate publish-subscribe interactions. Avoiding the need for direct reachability between clients, the broker only needs to be reachable from all clients. The broker also needs to have sufficient resources (storage, bandwidth, etc.) to host CoAP resource services, 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 REST API defined in this document. Clients initiate interactions with a 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 data 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. Every CoAP pub/sub topic has a link, consisting of a reference path on the broker using URI path [RFC3986] construction and link attributes [RFC6690]. Every topic is associated with zero or more stored representations with a content-format specified in the link. A CoAP pub/sub topic value may alternatively be a collection of one or more sub-topics, consisting of links to the sub-topic URIs and indicated by a link-format content-format.

3.5. Brokerless Pub/sub

Figure 2 shows an arrangement for using CoAP pub/sub in a "brokerless" configuration between peer nodes. Nodes in a brokerless system may act as both broker and client. The Broker interface in a brokerless node may be pre-configured with topics that expose services and resources. Brokerless peer nodes can be mixed with client and broker nodes in a system with full interoperability.

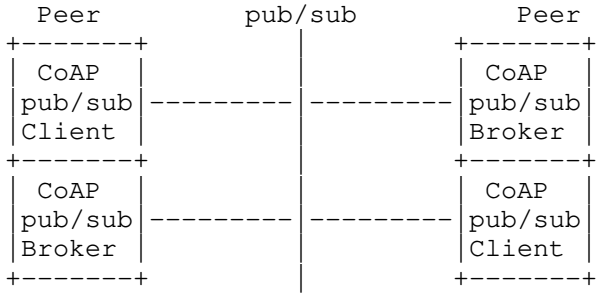


Figure 2: Brokerless pub/sub

4. CoAP Pub/sub REST API

This section defines the REST API exposed by a CoAP pub/sub Broker to pub/sub Clients. The examples throughout this section assume the use of CoAP [RFC7252]. A CoAP pub/sub Broker implementing this specification SHOULD support the DISCOVERY, CREATE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, READ, and REMOVE operations defined in this section. Optimized implementations MAY support a subset of the operations as required by particular constrained use cases.

4.1. DISCOVERY

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 the entry point of its pub/sub API at its .well-known/core location [RFC6690]. A CoAP pub/sub broker MAY register its pub/sub REST API entry point with a Resource Directory. Figure 3 shows an example of a client discovering a local pub/sub API using CoAP Simple Discovery. A broker wishing to advertise the CoAP pub/sub API for Simple Discovery or through a Resource Directory MUST use the link relation rt=core.ps. A broker MAY advertise its supported content formats and other attributes in the link to its pub/sub API.

A CoAP pub/sub Broker MAY offer a topic discovery entry point to enable Clients to find topics of interest, either by topic name or by link attributes which may be registered when the topic is created. Figure 4 shows an example of a client looking for a topic with a resource type (rt) of "temperature" using Discover. The client then receives the URI of the resource and its content-format. A pub/sub broker wishing to advertise topic discovery MUST use the relation rt=core.ps.discover in the link.

A CoAP pub/sub Broker MAY expose the Discover interface through the .well-known/core resource. Links to topics may be exposed at .well-known/core in addition to links to the pub/sub API. Figure 5 shows an example of topic discovery through .well-known/core.

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: application/link-format

The following response codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the broker resource. A pub/sub broker SHOULD use the value "/ps/" for the base URI of the pub/sub API wherever possible.

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

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

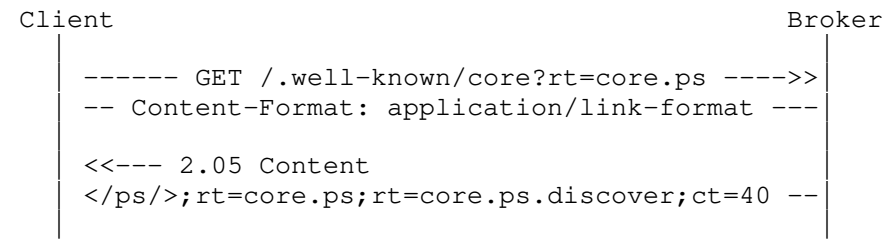


Figure 3: Example of DISCOVER pub/sub function

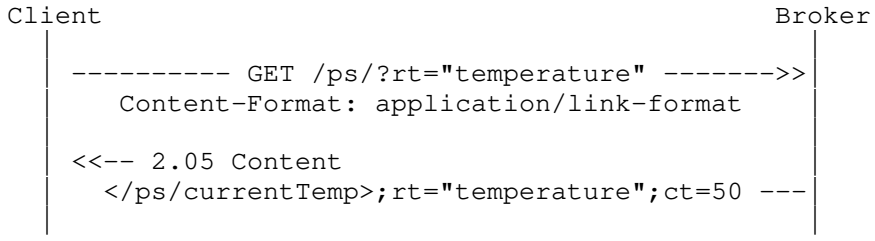


Figure 4: Example of DISCOVER topic

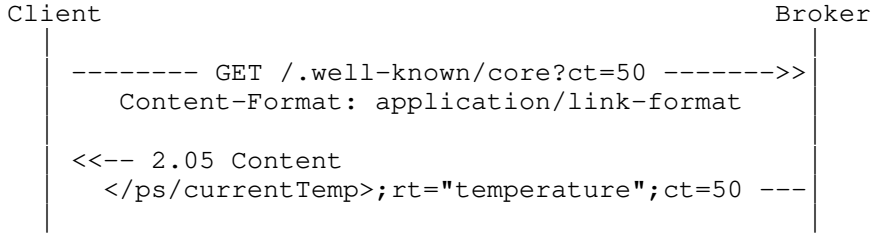


Figure 5: Example of DISCOVER topic

4.2. CREATE

A CoAP pubsub broker SHOULD allow Clients to create new topics on the broker using CREATE. Some exceptions are for fixed brokerless devices and pre-configured brokers in dedicated installations. A client wishing to create a topic MUST use CoAP POST to the pubsub API with a payload indicating the desired topic. The topic specification sent in the payload MUST use a supported serialization of the CoRE link format [RFC6690]. The target of the link MUST be a URI formatted string. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Format) link attribute in the link-format payload. The client MAY indicate the lifetime of the topic by including the Max-Age option in the CREATE request.

A Broker MUST return a response code of "2.01 Created" if the topic is created and return the URI path of the created topic via Location-Path options. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if a new topic can not be created. Broker SHOULD remove topics if the Max-Age of the topic is exceeded without any publishes to the topic. Broker SHOULD retain a topic indefinitely if the Max-Age option is elided or is set to zero upon topic creation. The lifetime of a topic MUST be refreshed upon create operations with a target of an existing topic.

Topics may be created as sub-topics of other topics. A client MAY create a topic with a ct (Content Format) link attribute value which describes a supported serialization of the CoRE link format [RFC6690] such as application/link-format (ct=40) or its JSON or CBOR serializations. If a topic is created which describes a link serialization, that topic may then have sub-topics created under it as shown in Figure 7.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for this interface:

Success: 2.01 "Created". Successful Creation of the topic

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.03 "Forbidden". Topic already exists.

Failure: 4.06 "Not Acceptable". Unsupported content format for topic.

Figure 6 shows an example of a topic called "topic1" being successfully created.

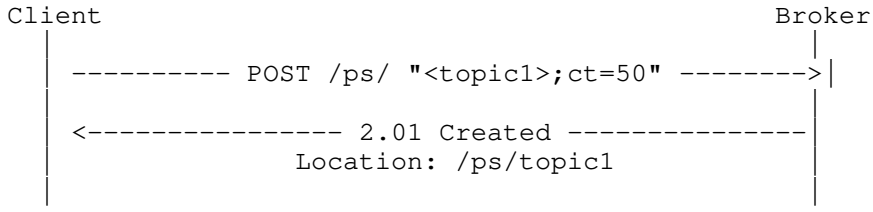


Figure 6: Example of CREATE topic

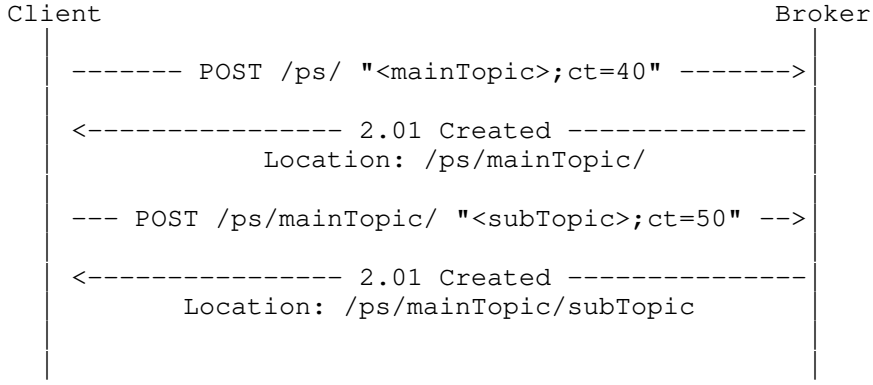


Figure 7: Example of CREATE sub-topic

4.3. PUBLISH

A CoAP pub/sub broker MAY allow clients to PUBLISH to topics on the broker. A client MAY use the PUT or the POST method to publish state updates to the CoAP pub/sub Broker. A client MUST use the content format specified upon creation of a given topic to publish updates to that topic. The broker MUST reject publish operations which do not use the specified content format. A CoAP client publishing on a topic MAY indicate the maximum lifetime of the value by including the Max-Age option in the publish request. The broker MUST return a response code of "2.04 Changed" if the publish is accepted. A Broker MAY return a "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.

A Broker MUST accept PUBLISH operations using the PUT method. PUBLISH operations using the PUT method replace any stored representation associated with the topic, with the supplied representation. A Broker MAY reject, or delay responses to, PUT

requests to a topic while pending resolution of notifications to subscribers from previous PUT requests.

Create on PUBLISH: A Broker MAY accept PUBLISH operations to new topics using the PUT method. If a Broker accepts a PUBLISH using PUT to a topic that does not exist, the Broker MUST create the topic using the information in the PUT operation. The Broker MUST create a topic with the URI-Path of the request, including all of the sub-topics necessary, and create a topic link with the ct attribute set to the content-format of the payload of the PUT request. If topic is created, the Broker MUST return the response "2.01 Created" with the URI of the created topic, including all of the created path segments, returned via the Location-Path option.

A Broker MAY accept PUBLISH operations using the POST method. If a broker accepts PUBLISH using POST it shall respond with the 2.04 Changed status code.

A Broker MAY perform garbage collection of stored representations which have been delivered to all subscribers or which have timed out. A Broker MAY retain at least one most recently published representation to return in response to SUBSCRIBE and READ requests.

A Broker MUST make a best-effort attempt to notify all clients subscribed on a particular topic each time it receives a publish on that topic. An example is shown in Figure 10. If a client publishes to a broker with the Max-Age option, the broker MUST include the same value for the Max-Age option in all notifications. A broker MUST use CoAP Notification as described in [RFC7641] to notify subscribed clients.

The PUBLISH interface is specified as follows:

Interaction: Client -> Broker

Method: PUT, POST

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: 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: 2.01 "Created". Successful publish, topic is created

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

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 7).

Figure 8 shows an example of a new value being successfully published to the topic "topic1". See Figure 10 for an example of a broker forwarding a message from a publishing client to a subscribed client.

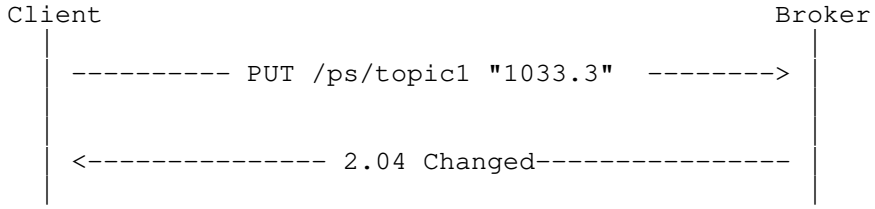


Figure 8: Example of PUBLISH

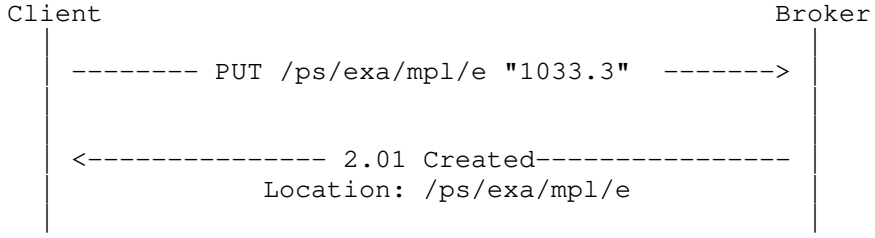


Figure 9: Example of CREATE on PUBLISH

4.4. SUBSCRIBE

A CoAP pub/sub broker MAY allow Clients to subscribe to topics on the Broker using CoAP Observe as described in [RFC7641]. A CoAP pub/sub Client wishing to Subscribe to a topic on a broker MUST use a CoAP GET with the Observe option set to 0 (zero). 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 reject Subscribe requests on a topic if the content format of the request is not supported by 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 "2.07 No Content" if the Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. The Broker MUST return a response code "4.15 Unsupported Content Format" if it can not return the requested content format. If a Broker is unable to accept a new Subscription on a topic, it SHOULD return the appropriate response code without the Observe option as per as per [RFC7641] Section 4.1. There is no explicit maximum lifetime of a Subscription, thus a Broker may remove subscribers at any time. The Broker, upon removing a Subscriber, will transmit the appropriate response code without the Observe option, as per [RFC7641] Section 4.2, to the removed Subscriber.

The SUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

The following response codes are defined for this interface:

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

Success: 2.07 "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 10 shows an example of Client2 subscribing to "topic1" and receiving a response from the broker, with a subsequent notification. The subscribe response from the broker uses the last stored value associated with the topic1. The notification from the broker is sent in response to the publish received from Client1.

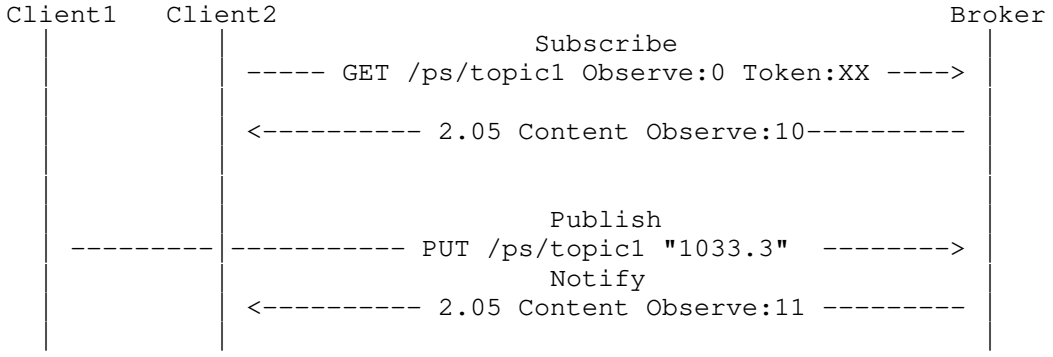


Figure 10: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

If a CoAP pub/sub broker allows clients to SUBSCRIBE to topics on the broker, it MUST allow Clients to unsubscribe from topics on the Broker using the CoAP Cancel Observation operation. A CoAP pub/sub Client wishing to unsubscribe to a topic on a Broker MUST either use

CoAP GET with Observe using an Observe parameter of 1 or send a CoAP Reset message in response to a publish, as per [RFC7641].

The UNSUBSCRIBE interface is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

The following response codes are defined for this interface:

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

Success: 2.07 "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 11 shows an example of a client unsubscribe using the Observe=1 cancellation method.

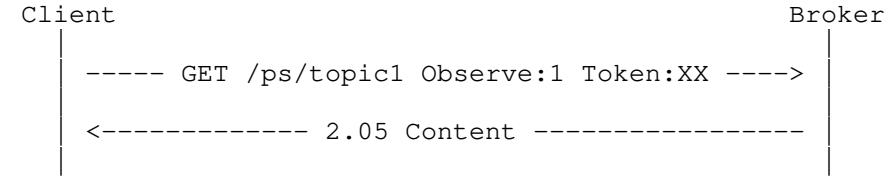


Figure 11: Example of UNSUBSCRIBE

4.6. READ

A CoAP pub/sub broker MAY accept Read requests on a topic using the the CoAP GET method 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 "2.07 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}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

The following response codes are defined for this interface:

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

Success: 2.07 "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 12 shows an example of a successful READ from topic1, followed by a Publish on the topic, followed at some time later by a read of the updated value from the recent Publish.

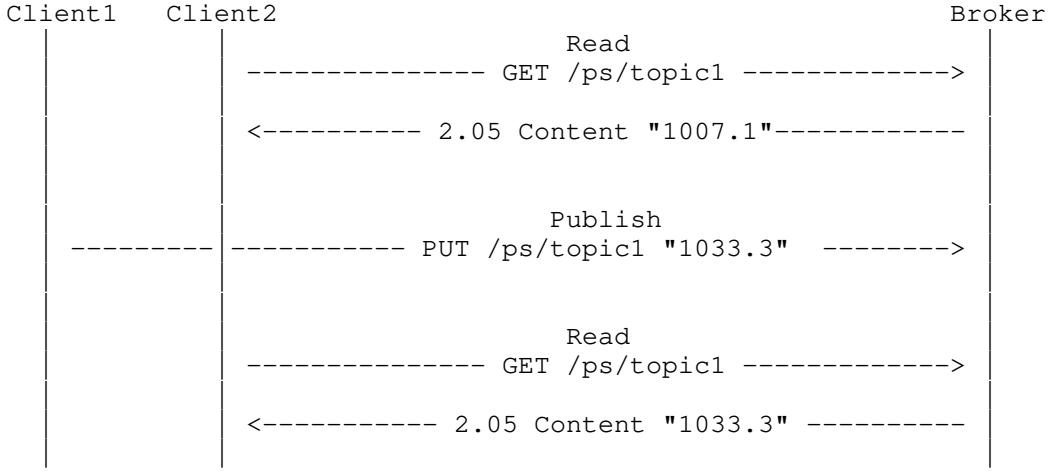


Figure 12: Example of READ

4.7. REMOVE

A CoAP pub/sub broker MAY allow clients to remove topics from the broker using the CoAP Delete method on the URI of the topic. The CoAP pub/sub Broker MUST return "2.02 Deleted" if the removal is successful. The broker MUST return the appropriate 4.xx response code indicating the reason for failure if the topic can not be removed. When a topic is removed for any reason, the Broker SHOULD return the response code 4.04 Not Found and remove all of the observers from the list of observers as per as per [RFC7641] Section 3.2. If a topic which has sub-topics is removed, then all of its sub-topics MUST be recursively removed.

The REMOVE interface is specified as follows:

Interaction: Client -> Broker

Method: DELETE

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: 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 13 shows a successful remove of topic1.

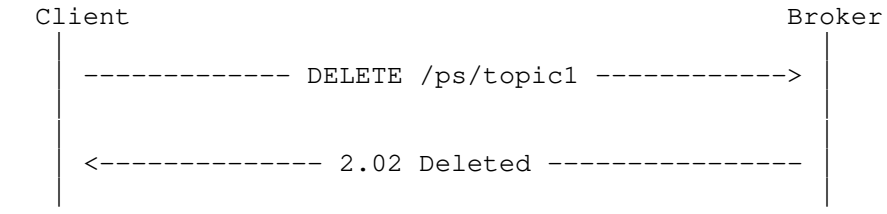


Figure 13: Example of REMOVE

5. CoAP Pub/sub Operation with Resource Directory

A CoAP pub/sub Broker may register the base URI, which is the REST API entry point for a pub/sub service, 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 links [RFC6690] with a Resource Directory to enable discovery of created pub/sub topics. 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.

A CoAP pub/sub Broker may alternatively register links to its topics to a Resource Directory by triggering the RD to retrieve it's links from .well-known/core. In order to use this method, the links must first be exposed in the .well-known/core of the pub/sub broker. See Section 4.1 in this document.

The pub/sub broker triggers the RD to retrieve its links by sending a POST with an empty payload to the .well-known/core of the Resource Directory. The RD server will then retrieve the links from the .well-known/core of the pub/sub broker and incorporate them into the Resource Directory. See [I-D.ietf-core-resource-directory] for further details.

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 SHOULD respond with Response Code 4.29, "Too Many Requests" [I-D.keranen-core-too-many-reqs] and set the Max-Age Option to indicate the number of seconds after which the client can 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 as described in [I-D.palombini-ace-coap-pubsub-profile]. When only end-to-end encryption is necessary and the CoAP Broker is trusted, Payload Only Protection (Mode:PAYL) could be used. The Publisher would wrap only the payload before sending it to the broker and set the option Content-Format to application/smpayl. Upon

receival, the Broker can read the unencrypted CoAP header to forward it to the subscribers.

9. IANA Considerations

This document registers one attribute value in the Resource Type (rt=) registry established with [RFC6690] and appends to the definition of one CoAP Response Code in the CoRE Parameters Registry.

9.1. Resource Type value 'core.ps'

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

9.2. Resource Type value 'core.ps.discover'

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

9.3. Response Code value '2.07'

- Response Code: 2.07
- Description: No Content
- Reference: [[This document]]
- Notes: The server sends this code to the client to indicate that the request was valid and accepted, but the response may contain an empty payload. It is comparable to and may be proxied with the HTTP 204 No Content status code.

10. Acknowledgements

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

11. References

11.1. Normative References

[I-D.keranen-core-too-many-reqs]
Keranen, A., "Too Many Requests Response Code for the Constrained Application Protocol", draft-keranen-core-too-many-reqs-00 (work in progress), March 2018.

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

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

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

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

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

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

11.2. Informative References

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-09 (work in progress), March 2018.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-13 (work in progress), March 2018.

[I-D.palombini-ace-coap-pubsub-profile]
Palombini, F., "CoAP Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-palombini-ace-coap-pubsub-profile-02 (work in progress), March 2018.

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

Authors' Addresses

Michael Koster
SmartThings

Email: Michael.Koster@smartthings.com

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson

Email: jaime.jimenez@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: August 25, 2018

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

February 21, 2018

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

Abstract

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

This specification defines more advanced, but still simple CoRE Congestion Control mechanisms, called CoCoA. The core of these mechanisms is a Retransmission TimeOut (RTO) algorithm that makes use of Round-Trip Time (RTT) estimates, in contrast with how the RTO is determined as per the base CoAP specification (RFC 7252). The mechanisms defined in this document have relatively low complexity, yet they improve the default CoAP RTO algorithm. The design of the mechanisms in this specification has made use of input from simulations and experiments in real networks.

This Internet-Draft will expire on August 25, 2018.

Table of Contents

1. Introduction 3
1.1. Terminology 3
2. Context 4
3. Area of Applicability 4
4. Advanced CoAP Congestion Control: RTO Estimation 5
4.1. Blind RTO Estimate 6
4.2. Measurement-based RTO Estimate 6
4.2.1. Differences with the algorithm of RFC 6298 7
4.2.2. Discussion 7
4.3. Lifetime, Aging 8
5. Advanced CoAP Congestion Control: Non-Confirmables 9
5.1. Discussion 9
6. IANA Considerations 9
7. Security Considerations 10
8. References 10
8.1. Normative References 10
8.2. Informative References 11
Appendix A. Supporting evidence 11
A.1. Older versions of the draft and improvement 12
A.2. References 12
Appendix B. Pseudocode 13
B.1. Updating the RTO estimator 13
B.2. RTO aging 14
B.3. Variable Backoff Factor 14
Appendix C. Examples 15
C.1. Example A.1: weak RTTs 15
C.2. Example A.2: VBF and aging 15
C.3. Example B: VBF and aging 16
Appendix D. Analysis: difference between strong and weak

estimators 16
Acknowledgements 17
Authors' Addresses 17

1. Introduction

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

The present specification defines such an advanced CoRE Congestion Control mechanism, with the goal of improving performance while retaining safety as well as the simplicity that is appropriate for constrained devices. Hence, we are calling this mechanism Simple Congestion Control/Advanced, or CoCoA for short.

CoCoA calculates the retransmission time-out (RTO) based on RTT estimations with and without loss. By taking retransmissions (in a potentially lossy network) into account when estimating the RTT, this algorithm reacts to congestion with a lower sending rate. For non-confirmable packets, it also limits the sending rate to 1/RTO; assuming that the RTO estimation in CoCoA works as expected, RTO should be slightly greater than the RTT, thus CoCoA would be more conservative than the original specification in [RFC7641].

In the Internet, congestion control is typically implemented in a way that it can be introduced or upgraded unilaterally. Still, a new congestion control scheme must not be introduced lightly. To ensure that the new scheme is not posing a danger to the network, considerable work has been done on simulations and experiments in real networks. Some of this work will be mentioned in "Discussion" subsections in the following sections; an overview is given in Appendix A. Extended rationale for this specification can also be found in the historical Internet-Drafts [I-D.bormann-core-congestion-control] and [I-D.eggert-core-congestion-control], as well as in the minutes of the IETF 84 CoRE WG meetings.

1.1. Terminology

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

Initiator: The endpoint that sends the message that initiates an exchange. E.g., the party that sends a confirmable message, or a non-confirmable message (see Section 4.3 of [RFC7252]) conveying a request.

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

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

2. Context

In the definition of the CoAP protocol [RFC7252], an approach was taken that includes a very simple basic scheme (lock-step with the number of parallel exchanges usually limited to 1) in the base specification together with performance-enhancing advanced mechanisms.

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

The present specification also assumes that, outside of exchanges, non-confirmable messages can only be used at a limited rate without an advanced congestion control mechanism (this is mainly relevant for [RFC7641]). It is also intended to address the [RFC8085] guideline about combining congestion control state for a destination; and to clarify its meaning for CoAP using the definition of an endpoint.

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

3. Area of Applicability

The present algorithm is intended to be generally applicable. The objective is to be "better" than default CoAP congestion control in a number of characteristics, including achievable goodput for a given offered load, latency, and recovery from bursts, while providing more predictable stress to the network and the same level of safety from catastrophic congestion. The algorithm defined in this document is

intended to adapt to the current characteristics of any underlying network, and therefore is well suited for a wide range of network conditions, in terms of bandwidth, latency, load, loss rate, topology, etc. In particular, CoCoA has been found to perform well in scenarios with latencies ranging from the order of milliseconds to peaks of dozens of seconds, as well as in single-hop and multihop topologies. Link technologies used in existing evaluation work comprise IEEE 802.15.4, GPRS, UMTS and Wi-Fi (see Appendix A). CoCoA is also expected to work suitably across the general Internet. The algorithm does require three state variables per scope plus the state needed to do RTT measurements, so it may not be applicable to the most constrained devices (say, class 1 as per [RFC7228]).

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

4. Advanced CoAP Congestion Control: RTO Estimation

For an initiator that plans to make multiple requests to one destination endpoint, it may be worthwhile to make RTT measurements in order to compute a more appropriate RTO than the default initial timeout of 2 to 3 s. In particular, a wide spectrum of RTT values is expected in different types of networks where CoAP is used. Those RTTs range from several orders of magnitude below the default initial timeout to values larger than the default. The algorithm defined in this document is based on the algorithm for RTO estimation defined in [RFC6298], with appropriately extended default/base values, as proposed in Section 4.2.1. Note that such a mechanism must, during idle periods, decay RTO estimates that are shorter or longer than the default RTO estimate back to the default RTO estimate, until fresh measurements become available again, as proposed in Section 4.3.

RTT variability challenges RTO estimation. In TCP, delayed ACKs contribute to RTT variability, since this option adds a delay of up to 500 ms (typically, 200 ms) before an ACK is sent by a receiving TCP endpoint. However, one important consideration not relevant for TCP is the fact that a CoAP round-trip may include application processing time, which may be hard to predict, and may differ between different resources available at the same endpoint. Also, for communications with networks of constrained devices that apply radio duty cycling, large and variable round-trip times are likely to be observed. Servers will only trigger their early ACKs (with a non-piggybacked response to be sent later) based on the default timers, e.g. after 1 s. A client that has arrived at a RTO estimate shorter than 1 s SHOULD therefore use a larger backoff factor for retransmissions to avoid expending all of its retransmissions

(MAX_RETRANSMIT, see Section 4.2 of [RFC7252], normally 4) in the default interval of 2 to 3 s. The approach chosen for a mechanism with variable backoff factors is presented in Section 4.2.1.

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

4.1. Blind RTO Estimate

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

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

4.2. Measurement-based RTO Estimate

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

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

```
RTO := w_weak   * E_weak_   + (1 - w_weak)   * RTO       (1)

RTO := w_strong * E_strong_ + (1 - w_strong) * RTO       (2)
```

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

The default values for the corresponding weights, w_weak and w_strong, are 0.25 and 0.5, respectively. These values have been found to offer good performance in evaluations (see Appendix A). Pseudocode and examples for the overall RTO estimate presented are available in Appendix B.1 and Appendix C.1.

4.2.1. Differences with the algorithm of RFC 6298

This subsection presents three differences of the algorithm defined in this document with the one defined in [RFC6298]. The first two recommend new parameter settings. The third one is the variable backoff factor (VBF), which replaces RFC6298's simple exponential backoff that always multiplies the RTO by a factor of 2 when the RTO timer expires.

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

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

In order to avoid that exchanges with small initial RTOs (i.e. RTO estimate lower than 1 s) use up all retransmissions in a short interval of time, the RTO for a retransmission is multiplied by 3 for each retransmission as long as the RTO is less than 1 s.

On the other hand, to avoid exchanges with large initial RTOs (i.e., RTO estimate greater than 3 s) not being able to carry out all retransmissions within MAX_TRANSMIT_WAIT (normally 93 s), the RTO is multiplied only by 1.5 when RTO is greater than 3 s.

Pseudocode for the variable backoff factor is in Appendix B.3.

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

4.2.2. Discussion

In contrast to [RFC6298], this algorithm attempts to make use of ambiguous information from retransmissions. This is motivated by the high non-congestion loss rates expected in constrained node networks,

and the need to update the RTO estimators even in the presence of loss. This approach appears to contravene the mandate in Section 3.1.1 of [RFC8085] that "latency samples MUST NOT be derived from ambiguous transactions". However, those samples are not simply combined into the strong estimator, but are used to correct the limited knowledge that can be gained from the strong RTT measurements by employing an additional weak estimator. In fact, the weak estimator allows to better update the RTO estimator when mostly weak RTTs are available, either due to the lossy nature of links or due to congestion-induced losses. In the presence of the latter, and compared to a strong-only estimator (w_weak=0), spurious timeouts are avoided and the rate of retries is reduced, which allows to decrease congestion. Evidence that has been collected from experiments appears to support that the overall effect of using this data in the way described is beneficial (Appendix A).

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

4.3. Lifetime, Aging

The state of the RTO estimators for an endpoint SHOULD be kept as long as possible. If other state is kept for the endpoint (such as a DTLS connection), it is very strongly RECOMMENDED to keep the RTO state alive at least as long as this other state. In the absence of such other state, the RTO state SHOULD be kept at least long enough to avoid frequent returns to inappropriate initial values. For the default parameter set of Section 4.8 of [RFC7252], it is strongly RECOMMENDED to keep it for at least 255 s.

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

```
RTO := 1 s + (0.5 * RTO)
```

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

Pseudocode and examples for the aging mechanism presented are available in Appendix B.2 and in Appendix C.2.

5. Advanced CoAP Congestion Control: Non-Confirmables

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

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

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

1. Of any 16 consecutive messages towards this endpoint that aren't responses or acknowledgments, at least 2 of the messages must be confirmable.

2. An RTO as specified in Section 4 must be used for confirmable messages.

3. The packet rate of non-confirmable messages cannot exceed 1/RTO, where RTO is the overall RTO estimator value at the time the non-confirmable packet is sent.

5.1. Discussion

The mechanism defined above for non-confirmables is relatively conservative. More advanced versions of this algorithm could run a TFRC-style Loss Event Rate calculator [RFC5348] and apply the TCP equation to achieve a higher rate than 1/RTO.

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

6. IANA Considerations

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

7. Security Considerations

The security considerations of, e.g., [RFC5681], [RFC2914], and [RFC8085] apply. Some issues are already discussed in the security considerations of [RFC7252].

If a malicious node manages to prevent the delivery of some packets, a consequence will be an RTO increase, which will further reduce network performance. Note that this type of attack is not specific for CoCoA (and not even specific for CoAP), and many congestion control algorithms increase the RTO upon packet loss detection. While it is hard to prevent radio jamming, some mitigation for other forms of this type of attack is provided by network access control techniques. Also, the weak estimator in CoCoA increases the chances of obtaining RTT measurements in the presence of heavy packet losses, allowing to keep the RTO updated, which in turn allows recovery from a jamming attack in reasonable time.

8. References

8.1. Normative References

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

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <https://www.rfc-editor.org/info/rfc2914>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

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

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

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

8.2. Informative References

[Betzler2013]
Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "Congestion control in reliable CoAP communication", ACM MSWIM'13 p. 365-372, DOI 10.1145/2507924.2507954, 2013.

[Betzler2015]
Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "CoCoA+: an Advanced Congestion Control Mechanism for CoAP", Ad Hoc Networks Vol. 33 pp. 126-139, DOI 10.1016/j.adhoc.2015.04.007, October 2015.

[I-D.bormann-core-congestion-control]
Bormann, C. and K. Hartke, "Congestion Control Principles for CoAP", draft-bormann-core-congestion-control-02 (work in progress), July 2012.

[I-D.eggert-core-congestion-control]
Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <https://www.rfc-editor.org/info/rfc5348>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

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

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

Appendix A. Supporting evidence

(Editor's note: The references local to this appendix may need to be merged with those from the specification proper, depending on the discretion of the RFC editor.)

CoCoA has been evaluated by means of simulation and experimentation in diverse scenarios comprising different link layer technologies, network topologies, traffic patterns and device classes. The main overall evaluation result is that CoCoA consistently delivers a performance which is better than, or at least similar to, that of default CoAP congestion control. While the latter is insensitive to network conditions, CoCoA is adaptive and makes good use of RTT samples.

It has been shown over real GPRS and IEEE 802.15.4 mesh network testbeds that in these settings, in comparison to default CoAP, CoCoA increases throughput and reduces the time it takes for a network to process traffic bursts, while not sacrificing fairness. In contrast, other RTT-sensitive approaches such as Linux-RTO or Peak-Hopper-RTO may be too simple or do not adapt well to IoT scenarios, underperforming default CoAP under certain conditions [1]. On the other hand, CoCoA has been found to reduce latency in GPRS and WiFi setups, compared with default CoAP [2].

CoCoA performance has also been evaluated for non-confirmable traffic over emulated GPRS/UMTS links and over a real IEEE 802.15.4 mesh testbed. Results show that since CoCoA is adaptive, it yields better packet delivery ratio than default CoAP (which does not apply congestion control to non-confirmable messages) or Observe (which introduces congestion control that is not adaptive to network conditions) [3, 4].

A.1. Older versions of the draft and improvement

CoCoA has evolved since its initial draft version. Its core has remained mostly stable since draft-bormann-core-cocoa-02. The evolution of CoCoA has been driven by research work. This process, including evaluations of early versions of CoCoA, as well as improvement proposals that were finally incorporated in CoCoA, is reflected in published works [5-10].

A.2. References

[1] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoAP congestion control for the Internet of Things", IEEE Communications Magazine, July 2016.

[2] F. Zheng, B. Fu, Z. Cao, "CoAP Latency Evaluation", draft-zheng-core-coap-lantency-evaluation-00, 2016 (work in progress).

[3] A. Betzler, C. Gomez, I. Demirkol, "Evaluation of Advanced Congestion Control Mechanisms for Unreliable CoAP Communications", PE-WASUN, Cancun, Mexico, 2015.

[4] A. Betzler, J. Isern, C. Gomez, I. Demirkol, J. Paradells, "Experimental Evaluation of Congestion Control for CoAP Communications without End-to-End Reliability", Ad Hoc Networks, Volume 52, 1 December 2016, Pages 183-194.

[5] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "Congestion Control in Reliable CoAP Communication", 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM'13), Barcelona, Spain, Nov. 2013.

[6] A. Betzler, C. Gomez, I. Demirkol, M. Kovatsch, "Congestion Control for CoAP cloud services", 8th International Workshop on Service-Oriented Cyber-Physical Systems in Converging Networked Environments (SOCNE) 2014, Barcelona, Spain, Sept. 2014.

[7] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoCoA+: an advanced congestion control mechanism for CoAP", Ad Hoc Networks journal, 2015.

[8] Bhalerao, Rahul, Sridhar Srinivasa Subramanian, and Joseph Pasquale. "An analysis and improvement of congestion control in the CoAP Internet-of-Things protocol." 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE, 2016.

[9] I Jaervinen, L Daniel, M Kojo, "Experimental evaluation of alternative congestion control algorithms for Constrained Application Protocol (CoAP)", IEEE 2nd World Forum on Internet of Things (WF-IoT), 2015.

[10] Balandina, Ekaterina, Yevgeni Koucheryavy, and Andrei Gurtov. "Computing the retransmission timeout in coap." Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg, 2013. 352-362.

Appendix B. Pseudocode

B.1. Updating the RTO estimator

```
// Default values
ALPHA = 0.125 // RFC 6298
BETA = 0.25 // RFC 6298
W_STRONG = 0.5
W_WEAK = 0.25

updateRTO(retransmissions, RTT) {
  if (retransmissions == 0) {
    RTTVAR_strong = (1 - BETA) * RTTVAR_strong
                  + BETA * (RTT_strong - RTT);
    RTT_strong  = (1 - ALPHA) * RTT_strong + ALPHA * RTT;
    E_strong = RTT_strong  + 4 * RTTVAR_strong;
    RTO = W_STRONG * E_strong + (1 - W_STRONG) * RTO;
  } else if (retransmissions <= 2) {
    RTTVAR_weak = (1 - BETA) * RTTVAR_weak
                + BETA * (RTT_weak - RTT);
    RTT_weak  = (1 - ALPHA) * RTT_weak + ALPHA * RTT;
    E_weak = RTT_weak  + 1 * RTTVAR_weak;
    RTO = W_WEAK * E_weak + (1 - W_WEAK) * RTO
  }
}
```

B.2. RTO aging

```
checkAging() {
  clock_time difference = getCurrentTime() - lastUpdatedTime;

  if ((RTO < 1s) && (difference > (16 * RTO))) {
    RTO = 2 * RTO;
    lastUpdatedTime = getCurrentTime();
  } else if ((RTO > 3s) && (difference > (4 * RTO))) {
    RTO = 1s + 0.5 * RTO;
    lastUpdatedTime = getCurrentTime();
  }
}
```

B.3. Variable Backoff Factor

```
backOffRTO() {
  if (RTO < 1s) {
    RTO = RTO * 3;
  } else if (RTO > 3s) {
    RTO = RTO * 1.5;
  } else {
    RTO = RTO * 2;
  }
}
```

Appendix C. Examples

C.1. Example A.1: weak RTTs

A large network of sensor nodes that report periodical measurements is operating normally, without congestion. The nodes transmit their sensor readings via CON messages every 20 s in an asynchronous way towards a server located behind a gateway, obtaining strong RTT measurements (RTT 1.1 s, RTTVAR 0.1 s) that lead to the calculation of an RTO of 1.5 s (in average) in each node. In this mode of operation, no aging is applied, since the RTO is refreshed before the aging mechanism applies.

Suddenly, upon detection of a global event, the majority of sensor nodes start transmitting at a higher rate (every 5 s) to increase the resolution of the acquired data, which creates heavy congestion that leads to packet losses and an important increase of real RTT between the nodes and the server (RTT 2 s, RTTVAR 1 s). Due to the packet losses and spurious retransmissions (which can fuel congestion even more), many nodes are not able to update their RTO via strong RTT measurements, but they are able to obtain weak RTT measurements. A node with an initial RTO of 1.5 s would run into a retransmission, before obtaining an ACK (given the RTT of 2 s and that the ACK is not lost).

This weak RTT measurement would increase the overall RTO of the node to 1.875 s (RTO = 0.25 * 3 s + 0.75 * 1.5 s). Following the same calculus (and RTT/RTTVAR values), after obtaining another weak RTT, the RTO would increase to 2.156 s. At this point, the benefits of the weak RTT measurements are twofold:

1. Further spurious retransmissions are avoided as the RTO has increased above the real RTT.

2. The increase of RTOs across the whole network reduces the rate with which retransmissions are generated, decreasing the network congestion (which leads to an RTT and packet loss decrease).

C.2. Example A.2: VBF and aging

Assuming that the frequency of message generation is even higher (every 3 s) and the real RTT would further increase due to congestion, the RTO at some point would increase to 4 s. Since now the RTO is above 3 s, no longer a binary backoff is used to avoid the RTO growing too much in case of retransmissions. As the generation of data from the nodes ceases at some point (the network returns to a normal state), the aging mechanism would reduce the RTO automatically

(with an RTO of 4 s, after 16 s the RTO would be shifted to 3 s before a new RTT is measured).

C.3. Example B: VBF and aging

A network of nodes connected over 4G with an Internet service is calculating very small RTO values (0.3 s) and the nodes are transmitting CON messages every 1 s. Suddenly, the connection quality gets worse and the nodes switch to a more stable, yet slower connection via GPRS. As a result of this change, the nodes run into retransmissions, as the real RTT has increased above the calculated RTO.

Since the RTO is below 1 s, the Variable Backoff Factor increases the backoff values quickly to avoid spurious retransmissions (0.9 s first retry, 2.7 s second retry, etc.). Further, if due to the packet losses and increased delays in the network no new RTT measurements are obtained, the aging mechanism automatically increases the RTO (doubling it) after 3.8 s (16 * 0.3 s) to adapt better to the sudden changes of network conditions. Without the Variable Backoff Factor and the aging mechanism, the number of spurious retransmissions would be much higher and the RTO would be corrected more slowly.

Appendix D. Analysis: difference between strong and weak estimators

This section analyzes the difference between the strong and weak RTO estimators. If there is no congestion, assume a static RTT of R'. Then, E_strong_can be expressed as:

$$E\_strong\_ = R' + G,$$

since RTTVAR is reduced constantly by RTTVAR = RTTVAR * 3/4 (according to [RFC6298], and SRTT=R'), G would be dominant term in the max(G, K * RTTVAR) expression in the long run.

For the weak estimator: assume that the RTO setting converges to E_strong_ calculated above in the long run. If there is a packet loss, and an RTT is obtained for the first retransmission, then the weak RTT sample obtained by the weak estimator is:

$$RW' = R' + G + R'$$

Therefore, E_weak_ can be expressed as:

$$E\_weak\_ = RW' + \max(G, RW'/2) = 3 * R'$$

Acknowledgements

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

Michael Scharf did a review of CoAP congestion control issues that asked a lot of good questions. Several Transport Area representatives made further significant inputs this discussion during IETF84, including Lars Eggert, Michael Scharf, and David Black. Andrew McGregor, Eric Rescorla, Richard Kelsey, Ed Beroset, Jari Arkko, Zach Shelby, Matthias Kovatsch and many others provided very useful additions. Further reviews by Michael Scharf and Ingemar Johansson led to further improvements, including some more discussion in the appendices.

Authors from Universitat Politecnica de Catalunya have been supported in part by the Spanish Government's Ministerio de Economia y Competitividad through projects TEC2009-11453, TEC2012-32531, TEC2016-79988-P and FEDER.

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

Authors' Addresses

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

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

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

Email: august.betzler@i2cat.net

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

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

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

Email: ilker.demirkol@entel.upc.edu

CoRE
Internet-Draft
Intended status: Standards Track
Expires: June 4, 2018

M. Veillette, Ed.
Trilliant Networks Inc.
P. van der Stok, Ed.
consultant
A. Pelov
Acklio
A. Bierman
YumaWorks
December 01, 2017

CoAP Management Interface
draft-ietf-core-comi-02

Abstract

This document describes a network management interface for constrained devices and networks, called CoAP Management Interface (CoMI). The Constrained Application Protocol (CoAP) is used to access datastore and data node resources specified in YANG, or SMIv2 converted to YANG. CoMI uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction. CoMI extends the set of YANG based protocols, NETCONF and RESTCONF, with the capability to manage constrained devices and networks.

Note

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

Status of This Memo

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

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

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

This Internet-Draft will expire on June 4, 2018.

Table of Contents

1. Introduction 3
1.1. Terminology 4
2. CoMI Architecture 5
2.1. Major differences between RESTCONF and CoMI 6
2.2. Compression of YANG identifiers 7
2.3. Instance identifier 8
2.4. CBOR ordered map schematic 8
2.5. Content-Formats 8
3. Example syntax 11
4. CoAP Interface 12
5. CoMI Collection Interface 13
5.1. Using the 'k' Uri-Query option 14
5.2. Data Retrieval 15
5.2.1. Using the 'c' Uri-Query option 16
5.2.2. Using the 'd' Uri-Query option 16
5.2.3. GET 17
5.2.4. FETCH 19
5.3. Data Editing 20
5.3.1. Data Ordering 20
5.3.2. POST 20
5.3.3. PUT 21
5.3.4. iPATCH 22
5.3.5. DELETE 23
5.4. Full datastore access 23
5.4.1. Full datastore examples 24
5.5. Event stream 25
5.5.1. Notify Examples 26
5.6. RPC statements 26
5.6.1. RPC Example 27
6. Access to MIB Data 27
7. Use of Block 29

8. Resource Discovery 29
9. Error Handling 31
10. Security Considerations 34
11. IANA Considerations 34
11.1. Resource Type (rt=) Link Target Attribute Values Registry 34
11.2. CoAP Content-Formats Registry 35
11.3. Media Types Registry 35
11.4. Concise Binary Object Representation (CBOR) Tags Registry 37
12. Acknowledgements 37
13. References 37
13.1. Normative References 38
13.2. Informative References 39
Appendix A. ietf-comi YANG module 40
Appendix B. ietf-comi .sid file 46
Appendix C. YANG example specifications 49
C.1. ietf-system 49
C.2. server list 51
C.3. interfaces 51
C.4. Example-port 52
C.5. IP-MIB 53
Authors' Addresses 55

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy, smart city 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. Messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

This draft describes the CoAP Management Interface which uses CoAP methods to access structured data defined in YANG [RFC7950]. This draft is complementary to [RFC8040] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data models specified in a standardized language, such as YANG, promotes interoperability between devices and applications from different manufacturers.

CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips.

To promote small messges, CoMI uses a YANG to CBOR mapping [I-D.ietf-core-yang-cbor] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length.

1.1. Terminology

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

The following terms are defined in the YANG data modelling language [RFC7950]: action, anydata, anyxml, client, configuration data, container, data model, data node, datastore, identity, instance identifier, key, key leaf, leaf, leaf-list, list, module, RPC, schema node, server, state data, submodule.

The following term is defined in [I-D.ietf-core-yang-cbor]: YANG schema item identifier (SID).

The following terms are defined in the CoAP protocol [RFC7252]: Confirmable Message, Content-Format.

The following terms are defined in this document:

data node resource: a CoAP resource that models a YANG data node.

datastore resource: a CoAP resource that models a YANG datastore.

event stream resource: a CoAP resource used by clients to observe YANG notifications.

target resource: the resource that is associated with a particular CoAP request, identified by the request URI.

data node instance: An instance of a data node specified in a YANG module and stored in the server.

notification instance: An instance of a schema node of type notification, specified in a YANG module implemented by the server. The instance is generated in the server at the occurrence of the corresponding event and reported by an event stream.

list instance identifier: Handle used to identify a YANG data node that is an instance of a YANG "list" specified with the values of the key leaves of the list.

single instance identifier: Handle used to identify a specific data node which can be instantiated only once. This includes data

nodes defined at the root of a YANG module or data nodes defined within a container. This excludes data nodes defined within a list or any children of these data nodes.

instance identifier: List instance identifier or single instance identifier.

data node value: The value assigned to a data node instance. Data node values are serialized into the payload according to the rules defined in section 4 of [I-D.ietf-core-yang-cbor].

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for reading and modifying the content of datastore(s) used for the management of the instrumented node.

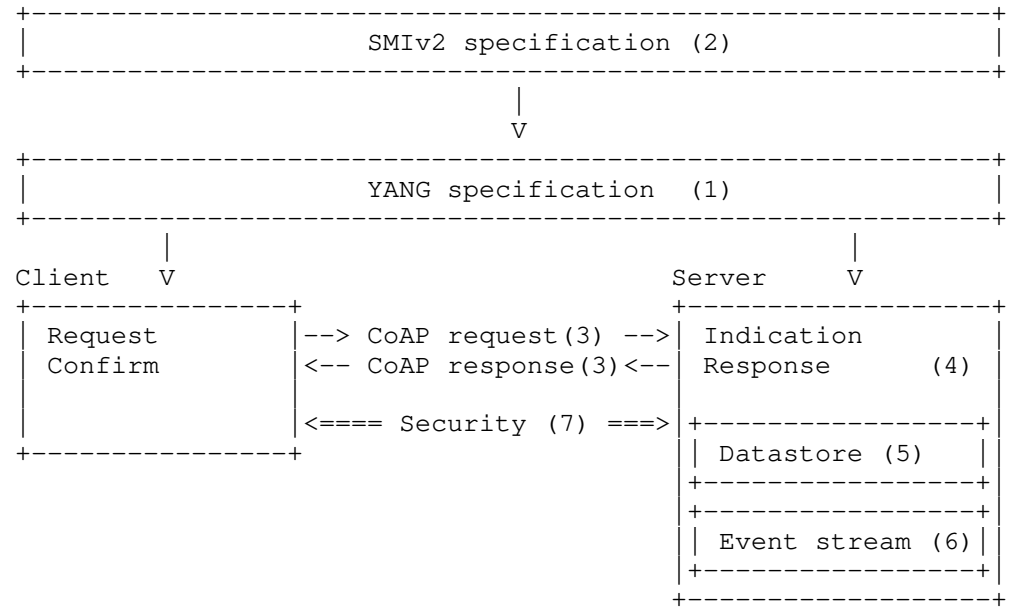


Figure 1: Abstract CoMI architecture

Figure 1 is a high-level representation of the main elements of the CoMI management architecture. The different numbered components of Figure 1 are discussed according to component number.

(1) YANG specification: contains a set of named and versioned modules.

(2) SMIv2 specification: A named module specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIv2 specifications to YANG specifications.

(3) CoAP request/response messages: The CoMI client sends request messages to and receives response messages from the CoMI server.

(4) Request, Indication, Response, Confirm: The processes performed by the CoMI clients and servers.

(5) Datastore: A resource used to access configuration data, state data, RPCs and actions. A CoMI server may support multiple datastores to support more complex operations such as configuration rollback, scheduled update.

(6) Event stream: An observable resource used to get real time notifications. A CoMI server may support multiple Event streams serving different purposes such as normal monitoring, diagnostic, syslog, security monitoring.

(7) Security: The server MUST prevent unauthorized users from reading or writing any CoMI resources. CoMI relies on security protocols such as DTLS [RFC6347] to secure CoAP communication.

2.1. Major differences between RESTCONF and CoMI

CoMI is a RESTful protocol for small devices where saving bytes to transport counts. Contrary to RESTCONF, many design decisions are motivated by the saving of bytes. Consequently, CoMI is not a RESTCONF over CoAP protocol, but differs more significantly from RESTCONF. Some major differences are cited below:

- CoMI uses CoAP/UDP as transport protocol and CBOR as payload format [I-D.ietf-core-yang-cbor]. RESTCONF uses HTTP/TCP as transport protocol and JSON [RFC7159] or XML [XML] as payload formats.

- CoMI encodes YANG identifier strings as numbers, where RESTCONF does not.

- CoMI uses the methods FETCH and iPATCH, not used by RESTCONF. RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not used by CoMI.

- CoMI does not support "insert" query parameter (first, last, before, after) and the "point" query parameter which are supported by RESTCONF.

- CoMI does not support the "start-time" and "stop-time" query parameters to retrieve past notifications.

- CoMI and RESTCONF also differ in the handling of:

  * notifications.

  * default values.

2.2. Compression of YANG identifiers

In the YANG specification, items are identified with a name string. In order to significantly reduce the size of identifiers used in CoMI, numeric identifiers are used instead of these strings. YANG Schema Item iDentifier (SID) is defined in [I-D.ietf-core-yang-cbor] section 2.1.

When used in a URI, SIDs are encoded in based64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5. The last 6 bits encoded is always aligned with the least significant 6 bits of the SID represented using an unsigned integer. 'A' characters (value 0) at the start of the resulting string are removed.

```
SID in basae64 = URLsafeChar[SID >> 60 & 0x3F] |
                 URLsafeChar[SID >> 54 & 0x3F] |
                 URLsafeChar[SID >> 48 & 0x3F] |
                 URLsafeChar[SID >> 42 & 0x3F] |
                 URLsafeChar[SID >> 36 & 0x3F] |
                 URLsafeChar[SID >> 30 & 0x3F] |
                 URLsafeChar[SID >> 24 & 0x3F] |
                 URLsafeChar[SID >> 18 & 0x3F] |
                 URLsafeChar[SID >> 12 & 0x3F] |
                 URLsafeChar[SID >> 6 & 0x3F] |
                 URLsafeChar[SID & 0x3F]
```

For example, SID 1721 is encoded as follow.

```
URLsafeChar[1721 >> 60 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 54 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 48 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 42 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 36 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 30 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 24 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 18 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 12 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 6 & 0x3F]  = URLsafeChar[26] = 'a'
URLsafeChar[1721 & 0x3F]       = URLsafeChar[57] = '5'
```

The resulting base64 representation of SID 1721 is "a5"

2.3. Instance identifier

Instance identifiers are used to uniquely identify data node instances within a datastore. This YANG built-in type is defined in [RFC7950] section 9.13. An instance identifier is composed of the data node identifier (i.e. a SID) and for data nodes within list(s) the keys used to index within these list(s).

When part of a payload, instance identifiers are encoded in CBOR based on the rules defined in [I-D.ietf-core-yang-cbor] section 5.13.1. When part of a URI, the SID is appended to the URI of the targeted datastore, the keys are specified using the 'k' URI-Query as defined in Section 5.1.

2.4. CBOR ordered map schematic

An ordered map is used as a root container of the application/yang-tree+cbor Content-Format. This datatype share the same functionalities as a CBOR map without the following limitations:

- The ordering of the pairs of data items is preserved from serialization to deserialization.
- Duplicate keys are allowed

This schematic is constructed using a CBOR array comprising pairs of data items, each pair consisting of a key that is immediately followed by a value. Unlike a CBOR map for which the length denotes the number of pairs, the length of the ordered map denotes the number of items (i.e. number of keys plus number of values).

The use of this schematic can be inferred from its context or by the presence of a preceding tag. The tag assigned to the Ordered map is defined in Section 11.4.

In the case of CoMI, the use of the ordered map as the root container of the application/yang-tree+cbor Content-Format is inferred, the Ordered map tag is not used.

2.5. Content-Formats

ComI uses Content-Formats based on the YANG to CBOR mapping specified in [I-D.ietf-core-yang-cbor]. All Content-Formats defined hereafter are constructed using one or both of these constructs:

- YANG data node value, encoded based on the rules defined in [I-D.ietf-core-yang-cbor] section 4.

- YANG instance identifier, encoded based on the rules defined in [I-D.ietf-core-yang-cbor] section 5.13.1.

The following Content-formats are defined:

application/yang-value+cbor: represents a CBOR YANG document containing one YANG data node value. The YANG data node instance can be a leaf, a container, a list, a list instance, a RPC input, a RPC output, an action input, an action output, a leaf-list, an anydata or an anyxml. The CBOR encoding for each of these YANG data node instances are defined in [I-D.ietf-core-yang-cbor] section 4.

FORMAT: data-node-value

DELTA ENCODING: SIDs included in a YANG container, a list instance, a RPC input, a RPC output, an action input, an actions output and an anydata are encoded using a delta value equal to the SID of the current schema node minus the SID of the parent. The parent SID of root data nodes is defined by the URI carried in the associated request (i.e. GET, PUT, POST).

application/yang-values+cbor: represents a YANG document containing a list of data node values.

FORMAT: CBOR array of data-node-value

DELTA ENCODING: SIDs included in a YANG container, a list instance and an anydata are encoded using a delta value equal to the SID of the current schema node minus the SID of the parent. The parent SID of root data nodes is defined by the corresponding instance-identifier carried in the FETCH request.

application/yang-tree+cbor: represents a CBOR YANG document containing a YANG data tree.

FORMAT: ordered map of single-instance-identifier, data-node-value

DELTA ENCODING: The SID part of the first instance-identifier within the ordered map is encoded using its absolute value. Subsequent instance-identifiers are encoded using a delta value equal to the SID of the current instance-identifiers minus the SID of the previous instance-identifier.

application/yang-selectors+cbor: represents a CBOR YANG document containing a list of data node selectors (i.e. instance identifier).

FORMAT: CBOR array of instance-identifier

DELTA ENCODING: The SID part of the first instance-identifier within the CBOR array is encoded using its absolute value. Subsequent instance-identifiers are encoded using a delta value equal to the SID of the current instance-identifiers minus the SID of the previous instance-identifier.

application/yang-patch+cbor: represents a CBOR YANG document containing a list of data nodes to be replaced, created, or deleted.

For each data node instance, D, for which the instance identifier is the same as for a data node instance, I, in the targeted resource: the data node value of D replaces the data node value of I. When the data node value of D is null, the data node instance I is removed. When the targeted resource does not contain a data node instance with the same instance identifier as D, a new data node instance is created in the targeted resource with the same instance identifier and data node value as D.

FORMAT: ordered map of instance-identifier, data-node-value

DELTA ENCODING: Same as Content-Format application/yang-tree+cbor

The different Content-formats usage is summarized in the table below:

Method	Resource	Content-Format
GET response	data node	/application/yang-value+cbor
PUT request	data node	/application/yang-value+cbor
POST request	data node	/application/yang-value+cbor
DELETE	data node	n/a
GET response	datastore	/application/yang-tree+cbor
PUT request	datastore	/application/yang-tree+cbor
POST request	datastore	/application/yang-tree+cbor
FETCH request	datastore	/application/yang-selectors+cbor
FETCH response	datastore	/application/yang-values+cbor
iPATCH request	datastore	/application/yang-patch+cbor
GET response	event stream	/application/yang-tree+cbor
POST request	rpc, action	/application/yang-value+cbor
POST response	rpc, action	/application/yang-value+cbor

3. Example syntax

This section presents the notation used for the examples. The YANG modules that are used throughout this document are shown in Appendix C. The example modules are copied from existing modules and annotated with SIDs. The values of the SIDs are taken over from [yang-cbor].

CBOR is used to encode CoMI request and response payloads. The CBOR syntax of the YANG payloads is specified in [RFC7049]. The payload examples are notated in Diagnostic notation (defined in section 6 of [RFC7049]) that can be automatically converted to CBOR.

SIDs in URIs are represented as a base64 number, SIDs in the payload are represented as decimal numbers.

4. CoAP Interface

The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Collection Interface. CoMI end-points that implement the CoMI management protocol, support at least one discoverable management resource of resource type (rt): core.c.datastore, with example path: /c, where c is short-hand for CoMI. The path /c is recommended but not compulsory (see Section 8).

Three CoMI resources are accessible with the following three example paths:

/c: Datastore resource with path "/c" and using CBOR content encoding format. Sub-resouces of format /c/instance-identifier may be available to access directly each data node resource for this datastore.

/mod.uri: URI identifying the location of the YANG module library used by this server, with path "/mod.uri" and Content-Format "text/plain; charset=utf-8". An ETag MUST be maintained for this resource by the server, which MUST be changed to a new value when the set of YANG modules in use by the server changes.

/s: Event stream resource to which YANG notification instances are reported. 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. Every data node of the YANG modules loaded in the CoMI server represents a sub-resource of the datastore resource (e.g. /c/instance-identifier).

When multiple instances of a list exist, instance selection is possible as described in Section 5.1, Section 5.2.4, and Section 5.2.3.1.

The description of the management collection interface, with if=core.c, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

Function	Recommended path	rt
Datastore	/c	core.c.datastore
Data node	/c/instance-identifier	core.c.datanode
YANG module library	/mod.uri	core.c.moduri
Event steam	/s	core.c.eventstream

The path values are example values. On discovery, the server makes the actual path values known for these four resources.

5. CoMI Collection Interface

The CoMI Collection Interface provides a CoAP interface to manage YANG servers.

The methods used by CoMI are:

Operation	Description
GET	Retrieve the datastore resource or a data node resource
FETCH	Retrieve specific data nodes within a datastore resource
POST	Create a datastore resource or a data node resource, invoke an RPC or action
PUT	Create or replace a datastore resource or a data node resource
iPATCH	Idem-potently create, replace, and delete data node resource(s) within a datastore resource
DELETE	Delete a datastore resource or a data node resource

There is one Uri-Query option for the GET, PUT, POST, and DELETE methods.

Uri-Query option	Description
k	Select an instance within YANG list(s)

This parameter is not used for FETCH and iPATCH, because their request payloads support list instance selection.

5.1. Using the 'k' Uri-Query option

The "k" (key) parameter specifies a specific instance of a data node. The SID in the URI is followed by the (?k=key1, key2,..). Where SID identifies a data node, and key1, key2 are the values of the key leaves that specify an instance. Lists can have multiple keys, and lists can be part of lists. The order of key value generation is given recursively by:

- For a given list, if a parent data node is a list, generate the keys for the parent list first.
- For a given list, generate key values in the order specified in the YANG module.

Key values are encoded using the rules defined in the following table.

YANG datatype	Uri-Query text content
uint8,uint16,unit32, uint64	int2str(key)
int8, int16,int32, int64	urlSafeBase64(CBORencode(key))
decimal64	urlSafeBase64(CBOR key)
string	key
boolean	"0" or "1"
enumeration	int2str(key)
bits	urlSafeBase64(CBORencode(key))
binary	urlSafeBase64(key)
identityref	int2str(key)
union	urlSafeBase64(CBORencode(key))
instance-identifier	urlSafeBase64(CBORencode(key))

In this table:

- The method int2str() is used to convert an integer value to a string. For example, int2str(0x0123) return the string "291".

- The method urlSafeBase64() is used to convert a binary string to base64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5. For example, urlSafeBase64(\xF9\x56\xA1\x3C) return the string "-VahPA".

- The method CBORencode() is used to convert a YANG value to CBOR as specified in [I-D.ietf-core-yang-cbor] section 5, item 8.

The resulting key string is encoded in a Uri-Query as specified in [RFC7252] section 6.5.

5.2. Data Retrieval

One or more data nodes can be retrieved by the client. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252] and to the FETCH method defined in section 2 of [RFC8132].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [RFC7959] may be used, as explained in more detail in Section 7.

There are two additional Uri-Query options for the GET and FETCH methods.

Uri-Query option	Description
c	Control selection of configuration and non-configuration data nodes (GET and FETCH)
d	Control retrieval of default values.

5.2.1. Using the 'c' Uri-Query option

The 'c' (content) parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
c	Return only configuration descendant data nodes
n	Return only non-configuration descendant data nodes
a	Return all descendant data nodes

This parameter is only allowed for GET and FETCH methods on datastore and data node resources. A 4.02 (Bad Option) error is returned if used for other methods or resource types.

If this Uri-Query option is not present, the default value is "a".

5.2.2. Using the 'd' Uri-Query option

The "d" (with-defaults) parameter controls how the default values of the descendant nodes of the requested data nodes will be processed.

The allowed values are:

Value	Description
a	All data nodes are reported. Defined as 'report-all' in section 3.1 of [RFC6243].
t	Data nodes set to the YANG default are not reported. Defined as 'trim' in section 3.2 of [RFC6243].

If the target of a GET or FETCH method is a data node that represents a leaf that has a default value, and the leaf has not been given a value by any client yet, the server MUST return the default value of the leaf.

If the target of a GET method is a data node that represents a container or list that has child resources with default values, and these have not been given value yet,

The server MUST not return the child resource if d= 't'

The server MUST return the child resource if d= 'a'.

If this Uri-Query option is not present, the default value is 't'.

5.2.3. GET

A request to read the values of a data node instance is sent with a confirmable CoAP GET message. An instance identifier is specified in the URI path prefixed with the example path /c.

FORMAT:
GET /c/instance-identifier

2.05 Content (Content-Format: application/yang-value+cbor)
data-node-value

The returned payload contains the CBOR encoding of the specified data node instance value.

5.2.3.1. GET Examples

Using for example the current-datetime leaf from Appendix C.1, a request is sent to retrieve the value of system-state/clock/current-datetime specified in container system-state. The SID of system-state/clock/current-datetime is 1723, encoded in octal 3273, yields two 6 bit decimal numbers 32 and 73, encoded in base64, (according to table 2 of [RFC4648]) yields a7. The response to the request returns

the CBOR encoding of this leaf of type 'string' as defined in [I-D.ietf-core-yang-cbor] section 5.4.

REQ: GET example.com/c/a3

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
"2014-10-26T12:16:31Z"

The next example represents the retrieval of a YANG container. In this case, the CoMI client performs a GET request on the clock container (SID = 1721; base64: a5). The container returned is encoded using a CBOR map as specified by [I-D.ietf-core-yang-cbor] section 4.2.

REQ: GET example.com/c/a5

RES: 2.05 Content (Content-Format: application/yang-value+cbor)

```
{
  +2 : "2014-10-26T12:16:51Z",   / current-datetime SID 1723 /
  +1 : "2014-10-21T03:00:00Z"    / boot-datetime SID 1722 /
}
```

This example shows the retrieval of the /interfaces/interface YANG list accessed using SID 1533 (base64: X9). The return payload is encoded using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1 containing 2 instances.

REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/yang-value+cbor)

```
[
  {
    +4 : "eth0",                / name  (SID 1537) /
    +1 : "Ethernet adaptor",    / description (SID 1534) /
    +5 : 1880,                  / type, (SID 1538) identity /
                                / ethernetCsmacd (SID 1880) /
    +2 : true                   / enabled ( SID 1535) /
  },
  {
    +4 : "eth1",                / name (SID 1537) /
    +1 : "Ethernet adaptor",    / description (SID 1534) /
    +5 : 1880,                  / type, (SID 1538) identity /
                                / ethernetCsmacd (SID 1880) /
    +2 : false                  / enabled /
  }
]
```

It is equally possible to select a leaf of a specific instance of a list. The example below requests the description leaf (SID=1534, base64: X-) within the interface list corresponding to the list key "eth0". The returned value is encoded in CBOR based on the rules specified by [I-D.ietf-core-yang-cbor] section 5.4.

REQ: GET example.com/c/X-?k="eth0"

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
"Ethernet adaptor"

5.2.4. FETCH

The FETCH is used to retrieve multiple data node values. The FETCH request payload contains a list of instance-identifier encoded based on the rules defined by Content-Format application/yang-selectors+cbor in Section 2.5. The return response payload contains a list of values encoded based on the rules defined by Content-Format application/yang-values+cbor in Section 2.5. A value MUST be returned for each instance-identifier specified in the request. A CBOR null is returned for each data node requested by the client, not supported by the server or not currently instantiated.

FORMAT:

```
  FETCH /c (Content-Format :application/yang-selectors+cbor)
  CBOR array of instance-identifier

  2.05 Content (Content-Format: application/yang-values+cbor)
  CBOR array of data-node-value
```

5.2.4.1. FETCH examples

The example uses the current-datetime leaf and the interface list from Appendix C.1. In the following example the value of current-datetime (SID 1723 and the interface list (SID 1533) instance identified with name="eth0" are queried.

```
REQ:  FETCH /c (Content-Format :application/yang-selectors+cbor)
[
  1723,            / current-datetime SID 1723 /
  [-190, "eth0"]   / interface SID 1533 with name = "eth0" /
]

RES:  2.05 Content (Content-Format :application/yang-value+cbor)
[
  "2014-10-26T12:16:31Z",       / current-datetime (SID 1723) /
  {
    +4 : "eth0",                / name (SID 1537) /
    +1 : "Ethernet adaptor",    / description (SID 1534) /
    +5 : 1880,                  / type (SID 1538), identity /
                                / ethernetCsmacd (SID 1880) /
    +2 : true                   / enabled (SID 1535) /
  }
]
```

5.3. Data Editing

CoMI allows datastore contents to be created, modified and deleted using CoAP methods.

5.3.1. Data Ordering

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as CBOR arrays so messages will preserve their order.

5.3.2. POST

The CoAP POST operation is used in CoMI for creation of data node resources and the invocation of "ACTION" and "RPC" resources. Refer to Section 5.6 for details on "ACTION" and "RPC" resources.

A request to create a data node resource is sent with a confirmable CoAP POST message. The URI specifies the data node to be instantiated at the exception of list intances. In this case, for compactness, the URI specifies the list for which an instance is created.

FORMAT:

```
  POST /c/<instance identifier>
  (Content-Format :application/yang-value+cbor)
  data-node-value

  2.01 Created
```

If the data node resource already exists, then the POST request MUST fail and a "4.09 Conflict" response code MUST be returned

5.3.2.1. Post example

The example uses the interface list from Appendix C.1. Example is creating a new list instance within the interface list (SID = 1533):

```
REQ: POST /c/X9 (Content-Format :application/yang-value+cbor)
{
  +4 : "eth5",              / name (SID 1537) /
  +1 : "Ethernet adaptor",  / description (SID 1534) /
  +5 : 1880,                / type (SID 1538), identity /
                            / ethernetCsmacd (SID 1880) /
  +2 : true                 / enabled (SID 1535) /
}


RES: 2.01 Created
```

5.3.3. PUT

A data node resource instance is created or replaced with the PUT method. A request to set the value of a data node instance is sent with a confirmable CoAP PUT message.

```
FORMAT:
  PUT /c/<instanceidentifier>
            (Content-Format :application/yang-value+cbor)
  data-node-value

  2.01 Created
```

5.3.3.1. PUT example

The example uses the interface list from Appendix C.1. Example is renewing an instance of the list interface (SID = 1533) with key name="eth0":

```
REQ: PUT /c/X9?k="eth0"
(Content-Format :application/yang-value+cbor)
{
  +4 : "eth0",             / name (SID 1537) /
  +1 : "Ethernet adaptor", / description (SID 1534) /
  +5 : 1880,               / type (SID 1538), identity /
                           / ethernetCsmacd (SID 1880) /
  +2 : true                / enabled (SID 1535) /
}

RES:  2.04 Changed
```

5.3.4. iPATCH

One or multiple data node instances are replaced with the idempotent iPATCH method [RFC8132]. A request is sent with a confirmable CoAP iPATCH message.

There are no Uri-Query options for the iPATCH method.

The processing of the iPATCH command is specified by Content-Format application/yang-patch+cbor. In summary, if the CBOR patch payload contains a data node instance that is not present in the target, this instance is added. If the target contains the specified instance, the content of this instance is replaced with the value of the payload. A null value indicates the removal of an existing data node instance.

FORMAT:

```
  iPATCH /c (Content-Format :application/yang-patch+cbor)
  ordered map of instance-identifier, data-node-value

  2.04 Changed
```

5.3.4.1. iPATCH example

In this example, a CoMI client requests the following operations:

- Set "/system/ntp/enabled" (SID 1755) to true.
- Remove the server "tac.nrc.ca" from the"/system/ntp/server" (SID 1756) list.
- Add the server "NTP Pool server 2" to the list "/system/ntp/server" (SID 1756).

```
REQ: iPATCH /c (Content-Format :application/yang-patch+cbor)
[
  1751 , true,                          / enabled (1755) /
  [+1, "tac.nrc.ca"], null,             / server (SID 1756) /
  +0,                                   / server (SID 1756) /
    {
      +3 : "tic.nrc.ca",                / name (SID 1759) /
      +4 : true,                        / prefer (SID 1760) /
      +5 : {                            / udp (SID 1761) /
        +1 : "132.246.11.231"           / address (SID 1762) /
      }
    }
]

RES: 2.04 Changed
```

5.3.5. DELETE

A data node resource is deleted with the DELETE method.

FORMAT:
  Delete /c/<instance identifier>

  2.02 Deleted

5.3.5.1. DELETE example

The example uses the interface list from Appendix C.3. Example is deleting an instance of the interface list (SID = 1533):

```
REQ:   DELETE /c/X9?k="eth0"

RES:   2.02 Deleted
```

5.4. Full datastore access

The methods GET, PUT, POST, and DELETE can be used to request, replace, create, and delete a whole datastore respectively.

FORMAT:
  GET /c

  2.05 Content (Content-Format: application/yang-tree+cbor)
  ordered map of single-instance-identifier, data-node-value

FORMAT:
PUT /c (Content-Format: application/yang-tree+cbor)
ordered map of single-instance-identifier, data-node-value

2.04 Changed

FORMAT:
POST /c (Content-Format: application/yang-tree+cbor)
ordered map of single-instance-identifier, data-node-value

2.01 Created

FORMAT:
DELETE /c

2.02 Deleted

The content of the ordered map represents the complete datastore of the server at the GET indication of after a successful processing of a PUT or POST request. When an Ordered map is used to carry a whole datastore, all data nodes MUST be identified using single instance identifiers (i.e. a SID), list instance identifiers are not allowed.

5.4.1. Full datastore examples

The example uses the interface list and the clock container from Appendix C.3. Assume that the datastore contains two modules ietf-system (SID 1700) and ietf-interfaces (SID 1500); they contain the list interface (SID 1533) with one instance and the container Clock (SID 1721). After invocation of GET, a map with these two modules is returned:

```
REQ:  GET /c

RES: 2.05 Content (Content-Format :application/yang-tree+cbor)
[
  1721,                           / Clock (SID 1721) /
    {
      +2: "2016-10-26T12:16:31Z", / current-datetime (SID 1723) /
      +1: "2014-10-05T09:00:00Z"  / boot-datetime (SID 1722) /
    },
  -188,                            / clock (SID 1533) /
    {
      +4 : "eth0",                / name (SID 1537) /
      +1 : "Ethernet adaptor",    / description (SID 1534) /
      +5 : 1880,                  / type (SID 1538), identity: /
                                  / ethernetCsmacd (SID 1880) /
      +2 : true                   / enabled (SID 1535) /
    }
]
```

5.5. Event stream

Event notification is an essential function for the management of servers. CoMI allows notifications specified in YANG [RFC5277] to be reported to a list of clients. The recommended path of the default event stream is /s. The server MAY support additional event stream resources to address different notification needs.

Reception of notification instances is enabled with the CoAP Observe [RFC7641] function. Clients subscribe to the notifications by sending a GET request with an "Observe" option, specifying the /s resource when the default stream is selected.

Each response payload carries one or multiple notifications. The number of notification reported and the conditions used to remove notifications from the reported list is left to the implementers. When multiple notifications are reported, they MUST be ordered starting from the newest notification at index zero.

An example implementation is:

> Every time an event is generated, the generated notification instance is appended to the chosen stream(s). After appending the instance, the content of the instance is sent to all clients observing the modified stream.
>
> Dependending on the storage space allocated to the notification stream, the oldest notifications that do not fit inside the notification stream storage space are removed.

```
FORMAT:
  Get /<stream-resource> Observe(0)

  2.05 Content (Content-Format :application/yang-tree+cbor)
  ordered map of instance-identifier, data-node-value
```

The array of data node instances may contain identical entries which have been generated at different times.

5.5.1. Notify Examples

Suppose the server generates the event specified in Appendix C.4. By executing a GET on the /s resource the client receives the following response:

```
REQ:  GET /s Observe(0) Token(0x93)

RES:  2.05 Content (Content-Format :application/yang-tree+cbor)
                        Observe(12) Token(0x93)
[
  60010,                  / example-port-fault (SID 60010) /
    {
      +1 : "0/4/21",      / port-name (SID 60011) /
      +2 : "Open pin 2"   / port-fault (SID 60012) /
    },
  +0,                     / example-port-fault (SID 60010) /
    {
      +1 : "1/4/21",      / port-name (SID 60011) /
      +2 : "Open pin 5"   / port-fault (SID 60012) /
    }
]
```

In the example, the request returns a success response with the contents of the last two generated events. Consecutively the server will regularly notify the client when a new event is generated.

To check that the client is still alive, the server MUST send confirmable notifications periodically. When the client does not confirm the notification from the server, the server will remove the client from the list of observers [RFC7641].

5.6. RPC statements

The YANG "action" and "RPC" statements specify the execution of a Remote procedure Call (RPC) in the server. It is invoked using a POST method to an "Action" or "RPC" resource instance. The request payload contains the values assigned to the input container when

specified. The response payload contains the values of the output container when specified. Both the input and output containers are encoded in CBOR using the rules defined in [I-D.ietf-core-yang-cbor] section 4.2.1. Root data nodes are encoded using the delta between the current SID and the SID of the invoked instance identifier a specified by the URI.

The returned success response code is 2.05 Content.

```
FORMAT:
  POST /c/<instance identifier>
            (Content-Format :application/yang-value+cbor)
  data-node-value

  2.05 Content (Content-Format :application/yang-value+cbor)
  data-node-value
```

5.6.1. RPC Example

The example is based on the YANG action specification of Appendix C.2. A server list is specified and the action "reset" (SID 60002, base64: Opq), that is part of a "server instance" with key value "myserver", is invoked.

```
REQ:  POST /c/Opq?k="myserver"
               (Content-Format :application/yang-value+cbor)
{
  +1 : "2016-02-08T14:10:08Z09:00" / reset-at (SID 60003) /
}

RES:  2.05 Content (Content-Format :application/yang-value+cbor)
{
  +2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (SID 60004)/
}
```

6. Access to MIB Data

Appendix C.5 shows a YANG module mapped from the SMI specification "IP-MIB" [RFC4293]. The following example shows the "ipNetToPhysicalEntry" list with 2 instances, using diagnostic notation without delta encoding.

```
{
   60021 :                    / list ipNetToPhysicalEntry /
   [
     {
       60022 : 1,             / ipNetToPhysicalIfIndex /
       60023 : 1,             / ipNetToPhysicalNetAddressType /
       60024 : h'0A000033',   / ipNetToPhysicalNetAddress /
       60025 : h'00000A01172D',/ ipNetToPhysicalPhysAddress /
       60026 : 2333943,       / ipNetToPhysicalLastUpdated /
       60027 : 4,             / ipNetToPhysicalType /
       60028 : 1,             / ipNetToPhysicalState /
       60029 : 1              / ipNetToPhysicalRowStatus /
     },
     {
       60022 : 1,             / ipNetToPhysicalIfIndex /
       60023 : 1,             / ipNetToPhysicalNetAddressType /
       60024 : h'09020304',   / ipNetToPhysicalNetAddress  /
       60025 : h'00000A36200A',/ ipNetToPhysicalPhysAddress /
       60026 : 2329836,       / ipNetToPhysicalLastUpdated /
       60027 : 3,             / ipNetToPhysicalType /
       60028 : 6,             / ipNetToPhysicalState /
       60029 : 1              / ipNetToPhysicalRowStatus /
     }
   ]
}
```

In this example one instance of /ip/ipNetToPhysicalEntry (SID 60021, base64: Oz1) that matches the keys ipNetToPhysicalIfIndex = 1, ipNetToPhysicalNetAddressType = ipv4 and ipNetToPhysicalNetAddress = 9.2.3.4 (h'09020304', base64: CQIDBA) is requested.

```
REQ: GET example.com/c/Oz1?k="1,1,CQIDBA"
```

```
RES: 2.05 Content (Content-Format: application/yang-value+cbor)
{
  +1 : 1,                  / ( SID 60022 ) /
  +2 : 1,                  / ( SID 60023 ) /
  +3 : h'09020304',        / ( SID 60024 ) /
  +4 : h'00000A36200A',    / ( SID 60025 ) /
  +5 : 2329836,            / ( SID 60026 ) /
  +6 : 3,                  / ( SID 60027 ) /
  +7 : 6,                  / ( SID 60028 ) /
  +8 : 1                   / ( SID 60029 ) /
}
```

7. Use of Block

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of data need to be transported, datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [RFC7959] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying transport sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

Beware of race conditions. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. On the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the resource, the transported results do not correspond with a state having occurred in the server; or worse the returned values are inconsistent. For example: array length does not correspond with the actual number of items. It may be advisable to use CBOR maps or CBOR arrays of undefined length, which are foreseen for data streaming purposes.

8. Resource Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.datastore" [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, the value "/c" is used as an example. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.c.datastore

RES: 2.05 Content
</c>; rt="core.c.datastore"
```

Implemented data nodes MAY be discovered using the standard CoAP resource discovery. The implementation can add the data node identifiers (SID) supported to /.well-known/core with

rt="core.c.datanode". The available SIDs can be discovered by sending a GET request to "/.well-known/core" including a resource type (rt) parameter with the value "core.c.datanode". Upon success, the return payload will contain the registered SIDs and their location.

The example below shows the discovery of the presence and location of data nodes.

REQ: GET /.well-known/core?rt=core.c.datanode

RES: 2.05 Content
</c/BaAiN>; rt="core.c.datanode",
</c/CF_fA>; rt="core.c.datanode"

The list of data nodes may become prohibitively long. Therefore, it is recommended to discover the details about the YANG modules implemented by reading a YANG module library (e.g. "ietf-comi-yang-library" ad defined by [I-D.veillette-core-yang-library]).

The resource "/mod.uri" is used to retrieve the location of the YANG module library. This library can be stored locally on each server, or remotely on a different server. The latter is advised when the deployment of many servers are identical.

The following example shows the URI of a local instance of container modules-state (SID=1802) as defined in [I-D.veillette-core-yang-library].

REQ: GET example.com/mod.uri

RES: 2.05 Content (Content-Format: text/plain; charset=utf-8)
example.com/c/cK

The following example shows the URI of a remote instance of same container.

REQ: GET example.com/mod.uri

RES: 2.05 Content (Content-Format: text/plain; charset=utf-8)
example-remote-server.com/group17/cK

Within the YANG module library all information about the module is stored such as: module identifier, identifier hierarchy, grouping, features and revision numbers.

9. Error Handling

In case a request is received which cannot be processed properly, the CoMI server MUST return an error message. This error message MUST contain a CoAP 4.xx or 5.xx response code.

Errors returned by a CoMI server can be broken into two categories, those associated to the CoAP protocol itself and those generated during the validation of the YANG data model constrains as described in [RFC7950] section 8.

The following list of common CoAP errors should be implemented by CoMI servers. This list is not exhaustive, other errors defined by CoAP and associated RFCs may be applicable.

- Error 4.01 (Unauthorized) is returned by the CoMI server when the CoMI client is not authorized to perform the requested action on the targeted resource (i.e. data node, datastore, rpc, action or event stream).

- Error 4.02 (Bad Option) is returned by the CoMI server when one or more CoAP options are unknown or malformed.

- Error 4.04 (Not Found) is returned by the CoMI server when the CoMI client is requesting a non-instantiated resource (i.e. data node, datastore, rpc, action or event stream).

- Error 4.05 (Method Not Allowed) is returned by the CoMI server when the CoMI client is requesting a method not supported on the targeted resource. (e.g. GET on an rpc, PUT or POST on a data node with "config" set to false).

- Error 4.08 (Request Entity Incomplete) is returned by the CoMI server if one or multiple blocks of a block transfer request is missing, see [RFC7959] for more details.

- Error 4.13 (Request Entity Too Large) may be returned by the CoMI server during a block transfer request, see [RFC7959] for more details.

- Error 4.15 (Unsupported Content-Format) is returned by the CoMI server when the Content-Format used in the request don't match those specified in section 2.3.

CoMI server MUST also enforce the different constraints associated to the YANG data models implemented. These constraints are described in [RFC7950] section 8. These errors are reported using the CoAP error code 4.00 (Bad Request) and may have the following error container as

payload. The YANG definition and associated .sid file are available in Appendix A and Appendix B. The error container is encoded using delta value equal to the SID of the current schema node minus the SID of the parent container (i.e 1024).

```
+--rw error!
   +--rw error-tag             identityref
   +--rw error-app-tag?        identityref
   +--rw error-data-node?      instance-identifier
   +--rw error-message?        string
```

The following error-tag and error-app-tag are defined by the ietf-comi YANG module, these tags are implemented as YANG identity and can be extended as needed.

- error-tag operation-failed is returned by the CoMI server when the operation request cannot be processed successfully.
  - error-app-tag malformed-message is returned by the CoMI server when the payload received from the CoMI client don't contain a well-formed CBOR content as defined in [RFC7049] section 3.3 or don't comply with the CBOR structure defined within this document.
  - error-app-tag data-not-unique is returned by the CoMI server when the validation of the 'unique' constraint of a list or leaf-list fails.
  - error-app-tag too-many-elements is returned by the CoMI server when the validation of the 'max-elements' constraint of a list or leaf-list fails.
  - error-app-tag too-few-elements is returned by the CoMI server when the validation of the 'min-elements' constraint of a list or leaf-list fails.
  - error-app-tag must-violation is returned by the CoMI server when the restrictions imposed by a 'must' statement are violated.
  - error-app-tag duplicate is returned by the CoMI server when a client tries to create a duplicate list or leaf-list entry.
- error-tag invalid-value is returned by the CoMI server when the CoMI client tries to update or create a leaf with a value encoded using an invalid CBOR datatype or if the 'range', 'length', 'pattern' or 'require-instance' constrain is not fulfilled.

  * error-app-tag invalid-datatype is returned by the CoMI server when CBOR encoding don't follow the rules set by or when the value is incompatible with the YANG Built-In type. (e.g. a value greater than 127 for an int8, undefined enumeration)

  * error-app-tag not-in-range is returned by the CoMI server when the validation of the 'range' property fails.

  * error-app-tag invalid-length is returned by the CoMI server when the validation of the 'length' property fails.

  * error-app-tag pattern-test-failed is returned by the CoMI server when the validation of the 'pattern' property fails.

- error-tag missing-element is returned by the CoMI server when the operation requested by a CoMI client fail to comply with the 'mandatory' constraint defined. The 'mandatory' constraint is enforced for leafs and choices, unless the node or any of its ancestors have a 'when' condition or 'if-feature' expression that evaluates to 'false'.

  * error-app-tag missing-key is returned by the CoMI server to further qualify an missing-element error. This error is returned when the CoMI client tries to create or list instance, without all the 'key' specified or when the CoMI client tries to delete a leaf listed as a 'key'.

  * error-app-tag missing-input-parameter is returned by the CoMI server when the input parameters of an RPC or action are incomplete.

- error-tag unknown-element is returned by the CoMI server when the CoMI client tries to access a data node of a YANG module not supported, of a data node associated to an 'if-feature' expression evaluated to 'false' or to a 'when' condition evaluated to 'false'.

- error-tag bad-element is returned by the CoMI server when the CoMI client tries to create data nodes for more than one case in a choice.

- error-tag data-missing is returned by the CoMI server when a data node required to accept the request is not present.

  * error-app-tag instance-required is returned by the CoMI server when a leaf of type 'instance-identifier' or 'leafref' marked with require-instance set to 'true' refers to an instance that does not exist.

  * error-app-tag missing-choice is returned by the CoMI server when no nodes exist in a mandatory choice.

- error-tag error is returned by the CoMI server when an unspecified error has occurred.

For example, the CoMI server might return the following error.

```
RES:  4.00 Bad Request (Content-Format :application/yang-value+cbor)
{
  +4 : 1011,       / error-tag (SID 1028) /
                   /   = invalid-value (SID 1011) /
  +1 : 1018,       / error-app-tag (SID 1025) /
                   /   = not-in-range (SID 1018) /
  +2 : 1740,       / error-data-node (SID 1026) /
                   /   = timezone-utc-offset (SID 1740) /
  +3 : "maximum value exceeded" / error-message (SID 1027) /
}
```

10. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. CoMI re-uses the security mechanisms already available to CoAP, this includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorization mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorization may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However, some adaptations may still be required, to cater for CoMI's specific requirements.

11. IANA Considerations

11.1. Resource Type (rt=) Link Target Attribute Values Registry

This document adds the following resource type to the "Resource Type (rt=) Link Target Attribute Values", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Value	Description	Reference
core.c.datastore	YANG datastore	RFC XXXX
core.c.datanode	YANG data node	RFC XXXX
core.c.liburi	YANG module library	RFC XXXX
core.c.eventstream	YANG event stream	RFC XXXX

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

11.2. CoAP Content-Formats Registry

This document adds the following Content-Format to the "CoAP Content-Formats", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Media Type	Excoding ID	Reference
application/yang-value+cbor	XXX	RFC XXXX
application/yang-values+cbor	XXX	RFC XXXX
application/yang-selectors+cbor	XXX	RFC XXXX
application/yang-tree+cbor	XXX	RFC XXXX
application/yang-ipatch+cbor	XXX	RFC XXXX

// RFC Ed.: replace XXX with assigned IDs and remove this note. // RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

11.3. Media Types Registry

This document adds the following media types to the "Media Types" registry.

Name	Template	Reference
yang-value+cbor	application/yang-value+cbor	RFC XXXX
yang-values+cbor	application/yang-values+cbor	RFC XXXX
yang-selectors+cbor	application/yang-selectors+cbor	RFC XXXX
yang-tree+cbor	application/yang-tree+cbor	RFC XXXX
yang-ipatch+cbor	application/yang-ipatch+cbor	RFC XXXX

Each of these media types share the following information:

- Subtype name: <as listed in table>
- Required parameters: N/A
- Optional parameters: N/A
- Encoding considerations: binary
- Security considerations: See the Security Considerations section of RFC XXXX
- Interoperability considerations: N/A
- Published specification: RFC XXXX
- Applications that use this media type: CoMI
- Fragment identifier considerations: N/A
- Additional information:

  * Deprecated alias names for this type: N/A
  * Magic number(s): N/A
  * File extension(s): N/A
  * Macintosh file type code(s): N/A

- Person & email address to contact for further information: iesg&ietf.org

- Intended usage: COMMON
- Restrictions on usage: N/A
- Author: Michel Veillette, ietf&augustcellars.com
- Change Controller: IESG
- Provisional registration? No

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

11.4. Concise Binary Object Representation (CBOR) Tags Registry

This document adds the following tags to the "Concise Binary Object Representation (CBOR) Tags" registry.

Tag	Data Item	Semantics	Reference
xxx	array	Oedered map	RFC XXXX

// RFC Ed.: replace xxx by the assigned Tag and remove this note. // RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

12. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification and specified CBOR encoding and use of hashes.

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Tanguy Ropitault, Juergen Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

13. References

13.1. Normative References

[I-D.ietf-core-sid]
Veillette, M. and A. Pelov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-02 (work in progress), October 2017.

[I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-05 (work in progress), August 2017.

[I-D.veillette-core-yang-library]
Veillette, M., "Constrained YANG Module Library", draft-veillette-core-yang-library-01 (work in progress), July 2017.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://www.rfc-editor.org/info/rfc5277>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <https://www.rfc-editor.org/info/rfc6243>.

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

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

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

13.2. Informative References

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-10 (work in progress), September 2017.

[netconfcentral]
YUMAworks, "NETCONF Central: library of YANG modules", Web http://www.netconfcentral.org/modulelist.

[RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <https://www.rfc-editor.org/info/rfc4293>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

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

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[XML] W3C, "Extensible Markup Language (XML)", Web http://www.w3.org/xml.

[yang-cbor] Veillette, M., "yang-cbor Registry", Web https://github.com/core-wg/yang-cbor/tree/master/registry/.

Appendix A. ietf-comi YANG module

```
<CODE BEGINS> file "ietf-comi@2017-07-01.yang"
module ietf-comi {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-comi";
  prefix comi;

  organization
    "IETF Core Working Group";

  contact
    "Michel Veillette
     <mailto:michel.veillette@trilliantinc.com>

     Alexander Pelov
     <mailto:alexander@ackl.io>

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

     Andy Bierman
     <mailto:andy@yumaworks.com>";

  description
    "This module contains the different definitions required
     by the CoMI protocol.";

  revision 2017-07-01 {
     description
      "Initial revision.";
```

```
    reference
      "[I-D.ietf-core-comi] CoAP Management Interface";
  }

  typedef sid {
    type uint64;
    description
      "YANG Schema Item iDentifier";
    reference
      "[I-D.ietf-core-sid] YANG Schema Item iDentifier (SID)";
  }

  typedef date_and_time_b {
    type int64;
    description
      "Binary representation of a date and time. This value is
       encoded using a positive or negative value representing
       a number of seconds relative to 1970-01-01T00:00Z in UTC
       time (i.e. the epoch). Negative values represent a date
       and time before the epoch, positive values represent a
       date and time after the epoch.
       This representation is defined in [RFC 7049] section
       2.4.1. When implemented using CoMI, tag 0 is assumed
       and not encoded.";
    reference
      "[RFC 7049] Concise Binary Object Representation (CBOR)";
  }

  identity error-tag {
    description
      "Base identity for error-tag.";
  }

  identity operation-failed {
    base error-tag;
    description
      "Returned by the CoMI server when the operation request
       can't be processed successfully.";
  }

  identity invalid-value {
    base error-tag;
    description
      "Returned by the CoMI server when the CoMI client tries to
       update or create a leaf with a value encoded using an
       invalid CBOR datatype or if the 'range', 'length',
       'pattern' or 'require-instance' constrain is not
       fulfilled.";
```

```
}

identity missing-element {
  base error-tag;
  description
    "Returned by the CoMI server when the operation requested
     by a CoMI client fails to comply with the 'mandatory'
     constraint defined. The 'mandatory' constraint is
     enforced for leafs and choices, unless the node or any of
     its ancestors have a 'when' condition or 'if-feature'
     expression that evaluates to 'false'.";
}

identity unknown-element {
  base error-tag;
  description
    "Returned by the CoMI server when the CoMI client tries to
     access a data node of a YANG module not supported, of a
     data node associated with an 'if-feature' expression
     evaluated to 'false' or to a 'when' condition evaluated
     to 'false'.";
}

identity bad-element {
  base error-tag;
  description
    "Returned by the CoMI server when the CoMI client tries to
     create data nodes for more than one case in a choice.";
}

identity data-missing {
  base error-tag;
  description
    "Returned by the CoMI server when a data node required to
     accept the request is not present.";
}

identity error {
  base error-tag;
  description
    "Returned by the CoMI server when an unspecified error has
    occurred.";
}

identity error-app-tag {
  description
    "Base identity for error-app-tag.";
}
```

```
identity malformed-message {
  base error-app-tag;
  description
    "Returned by the CoMI server when the payload received
     from the CoMI client don't contain a well-formed CBOR
     content as defined in [RFC7049] section 3.3 or don't
     comply with the CBOR structure defined within this
     document.";
}

identity data-not-unique {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
     'unique' constraint of a list or leaf-list fails.";
}

identity too-many-elements {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
     'max-elements' constraint of a list or leaf-list fails.";
}

identity too-few-elements {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
     'min-elements' constraint of a list or leaf-list fails.";
}

identity must-violation {
  base error-app-tag;
  description
    "Returned by the CoMI server when the restrictions
     imposed by a 'must' statement are violated.";
}

identity duplicate {
  base error-app-tag;
  description
    "Returned by the CoMI server when a client tries to create
     a duplicate list or leaf-list entry.";
}

identity invalid-datatype {
  base error-app-tag;
  description
```

```
        "Returned by the CoMI server when CBOR encoding is
         incorect or when the value encoded is incompatible with
         the YANG Built-In type. (e.g. value greater than 127
         for an int8, undefined enumeration).";
    }

    identity not-in-range {
      base error-app-tag;
      description
        "Returned by the CoMI server when the validation of the
         'range' property fails.";
    }

    identity invalid-length {
      base error-app-tag;
      description
        "Returned by the CoMI server when the validation of the
         'length' property fails.";
    }

    identity pattern-test-failed {
      base error-app-tag;
      description
        "Returned by the CoMI server when the validation of the
         'pattern' property fails.";
    }

    identity missing-key {
      base error-app-tag;
      description
        "Returned by the CoMI server to further qualify a
         missing-element error. This error is returned when the
         CoMI client tries to create or list instance, without all
         the 'key' specified or when the CoMI client tries to
         delete a leaf listed as a 'key'.";
    }

    identity missing-input-parameter {
      base error-app-tag;
      description
        "Returned by the CoMI server when the input parameters
         of a RPC or action are incomplete.";
    }

    identity instance-required {
      base error-app-tag;
      description
        "Returned by the CoMI server when a leaf of type
```

```
          'instance-identifier' or 'leafref' marked with
          require-instance set to 'true' refers to an instance
          that does not exist.";
      }

      identity missing-choice {
        base error-app-tag;
        description
          "Returned by the CoMI server when no nodes exist in a
           mandatory choice.";
      }

      container error {
        presence "Error paylaod";

        description
          "Optional payload of a 4.00 Bad Request CoAP error.";

        leaf error-tag {
          type identityref {
            base error-tag;
          }
          mandatory true;
          description
            "The enumerated error-tag.";
        }

        leaf error-app-tag {
          type identityref {
            base error-app-tag;
          }
          description
            "The application-specific error-tag.";
        }

        leaf error-data-node {
          type instance-identifier;
          description
            "When the error reported is caused by a specific data node,
             this leaf identifies the data node in error.";
        }

        leaf error-message {
          type string;
          description
            "A message describing the error.";
        }
      }
```

```
   }
   <CODE ENDS>
```

Appendix B. ietf-comi .sid file

```
   {
     "assignment-ranges": [
       {
         "entry-point": 1000,
         "size": 100
       }
     ],
     "module-name": "ietf-comi",
     "module-revision": "2017-07-01",
     "items": [
       {
         "namespace": "module",
         "identifier": "ietf-comi",
         "sid": 1000
       },
       {
         "namespace": "identity",
         "identifier": "bad-element",
         "sid": 1001
       },
       {
         "namespace": "identity",
         "identifier": "data-missing",
         "sid": 1002
       },
       {
         "namespace": "identity",
         "identifier": "data-not-unique",
         "sid": 1003
       },
       {
         "namespace": "identity",
         "identifier": "duplicate",
         "sid": 1004
       },
       {
         "namespace": "identity",
         "identifier": "error",
         "sid": 1005
       },
       {
         "namespace": "identity",
         "identifier": "error-app-tag",
```

```
      "sid": 1006
    },
    {
      "namespace": "identity",
      "identifier": "error-tag",
      "sid": 1007
    },
    {
      "namespace": "identity",
      "identifier": "instance-required",
      "sid": 1008
    },
    {
      "namespace": "identity",
      "identifier": "invalid-datatype",
      "sid": 1009
    },
    {
      "namespace": "identity",
      "identifier": "invalid-length",
      "sid": 1010
    },
    {
      "namespace": "identity",
      "identifier": "invalid-value",
      "sid": 1011
    },
    {
      "namespace": "identity",
      "identifier": "malformed-message",
      "sid": 1012
    },
    {
      "namespace": "identity",
      "identifier": "missing-choice",
      "sid": 1013
    },
    {
      "namespace": "identity",
      "identifier": "missing-element",
      "sid": 1014
    },
    {
      "namespace": "identity",
      "identifier": "missing-input-parameter",
      "sid": 1015
    },
    {
```

```
        "namespace": "identity",
        "identifier": "missing-key",
        "sid": 1016
      },
      {
        "namespace": "identity",
        "identifier": "must-violation",
        "sid": 1017
      },
      {
        "namespace": "identity",
        "identifier": "not-in-range",
        "sid": 1018
      },
      {
        "namespace": "identity",
        "identifier": "operation-failed",
        "sid": 1019
      },
      {
        "namespace": "identity",
        "identifier": "pattern-test-failed",
        "sid": 1020
      },
      {
        "namespace": "identity",
        "identifier": "too-few-elements",
        "sid": 1021
      },
      {
        "namespace": "identity",
        "identifier": "too-many-elements",
        "sid": 1022
      },
      {
        "namespace": "identity",
        "identifier": "unknown-element",
        "sid": 1023
      },
      {
        "namespace": "data",
        "identifier": "/ietf-comi:error",
        "sid": 1024
      },
      {
        "namespace": "data",
        "identifier": "/ietf-comi:error/error-app-tag",
        "sid": 1025
```

```
      },
      {
        "namespace": "data",
        "identifier": "/ietf-comi:error/error-data-node",
        "sid": 1026
      },
      {
        "namespace": "data",
        "identifier": "/ietf-comi:error/error-message",
        "sid": 1027
      },
      {
        "namespace": "data",
        "identifier": "/ietf-comi:error/error-tag",
        "sid": 1028
      }
    ]
  }
```

Appendix C. YANG example specifications

This appendix shows five YANG example specifications taken over from as many existing YANG modules. The YANG modules are available from [netconfcentral]. Each YANG item identifier is accompanied by its SID shown after the "//" comment sign.

C.1. ietf-system

Excerpt of the YANG module ietf-system [RFC7317].

```
module ietf-system {                    // SID 1700
  container system {                    // SID 1717
    container clock {                   // SID 1738
      choice timezone {
        case timezone-name {
          leaf timezone-name {          // SID 1739
            type timezone-name;
          }
        }
        case timezone-utc-offset {
          leaf timezone-utc-offset {    // SID 1740
            type int16 {
            }
          }
        }
      }
    }
    container ntp {                     // SID 1754
```

```
        leaf enabled {                    // SID 1755
          type boolean;
          default true;
        }
        list server {                     // SID 1756
          key name;
          leaf name {                     // SID 1759
            type string;
          }
          choice transport {
            case udp {
              container udp {             // SID 1761
                leaf address {            // SID 1762
                  type inet:host;
                }
                leaf port {               // SID 1763
                  type inet:port-number;
                }
              }
            }
          }
          leaf association-type {         // SID 1757
            type enumeration {
              enum server {
              }
              enum peer {
              }
              enum pool {
              }
            }
          }
          leaf iburst {                   // SID 1758
            type boolean;
          }
          leaf prefer {                   // SID 1760
            type boolean;
            default false;
          }
        }
      }
    container system-state {              // SID 1720
      container clock {                   // SID 1721
        leaf current-datetime {           // SID 1723
          type yang:date-and-time;
        }
        leaf boot-datetime {              // SID 1722
          type yang:date-and-time;
        }
```

```
      }
    }
  }
```

C.2. server list

Taken over from [RFC7950] section 7.15.3.

```
  module example-server-farm {
    yang-version 1.1;
    namespace "urn:example:server-farm";
    prefix "sfarm";

    import ietf-yang-types {
      prefix "yang";
    }

    list server {                          // SID 60000
      key name;
      leaf name {                          // SID 60001
        type string;
      }
      action reset {                       // SID 60002
        input {
          leaf reset-at {                  // SID 60003
            type yang:date-and-time;
            mandatory true;
          }
        }
        output {
          leaf reset-finished-at {         // SID 60004
            type yang:date-and-time;
            mandatory true;
          }
        }
      }
    }
  }
```

C.3. interfaces

Excerpt of the YANG module ietf-interfaces [RFC7223].

```
   module ietf-interfaces {                 // SID 1500
     container interfaces {                 // SID 1505
       list interface {                     // SID 1533
         key "name";
         leaf name {                        // SID 1537
           type string;
         }
         leaf description {                 // SID 1534
           type string;
         }
         leaf type {                        // SID 1538
           type identityref {
             base interface-type;
           }
           mandatory true;
         }

         leaf enabled {                     // SID 1535
           type boolean;
           default "true";
         }

         leaf link-up-down-trap-enable {    // SID 1536
           if-feature if-mib;
           type enumeration {
             enum enabled {
               value 1;
             }
             enum disabled {
               value 2;
             }
           }
         }
       }
     }
   }
```

C.4. Example-port

Notification example defined within this document.

```
   module example-port {
           ...
           notification example-port-fault {   // SID 60010
             description
               "Event generated if a hardware fault on a
                line card port is detected";
             leaf port-name {                  // SID 60011
               type string;
               description "Port name";
             }
             leaf port-fault {                 // SID 60012
               type string;
               description "Error condition detected";
             }
           }
         }
```

C.5. IP-MIB

The YANG translation of the SMI specifying the IP-MIB [RFC4293], extended with example SID numbers, yields:

```
   module IP-MIB {
     import IF-MIB {
       prefix if-mib;
     }
     import INET-ADDRESS-MIB {
       prefix inet-address;
     }
     import SNMPv2-TC {
       prefix smiv2;
     }
     import ietf-inet-types {
       prefix inet;
     }
     import yang-smi {
       prefix smi;
     }
     import ietf-yang-types {
       prefix yang;
     }

     container ip {                            // SID 60020
       list ipNetToPhysicalEntry {             // SID 60021
         key "ipNetToPhysicalIfIndex
              ipNetToPhysicalNetAddressType
              ipNetToPhysicalNetAddress";
         leaf ipNetToPhysicalIfIndex {         // SID 60022
```

```
          type if-mib:InterfaceIndex;
        }
        leaf ipNetToPhysicalNetAddressType {  // SID 60023
          type inet-address:InetAddressType;
        }
        leaf ipNetToPhysicalNetAddress {      // SID 60024
          type inet-address:InetAddress;
        }
        leaf ipNetToPhysicalPhysAddress {     // SID 60025
          type yang:phys-address {
            length "0..65535";
          }
        }
        leaf ipNetToPhysicalLastUpdated {     // SID 60026
          type yang:timestamp;
        }
        leaf ipNetToPhysicalType {            // SID 60027
          type enumeration {
            enum "other" {
              value 1;
            }
            enum "invalid" {
              value 2;
            }
            enum "dynamic" {
              value 3;
            }
            enum "static" {
              value 4;
            }
            enum "local" {
              value 5;
            }
          }
        }
        leaf ipNetToPhysicalState {           // SID 60028
          type enumeration {
            enum "reachable" {
              value 1;
            }
            enum "stale" {
              value 2;
            }
            enum "delay" {
              value 3;
            }
            enum "probe" {
              value 4;
```

```
            }
            enum "invalid" {
              value 5;
            }
            enum "unknown" {
              value 6;
            }
            enum "incomplete" {
              value 7;
            }
          }
        }
        leaf ipNetToPhysicalRowStatus {       // SID 60029
          type smiv2:RowStatus;
      }  // list ipNetToPhysicalEntry
    }  // container ip
  }  // module IP-MIB
```

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Email: michel.veillette@trilliantinc.com

Peter van der Stok (editor)
consultant

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

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

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

Email: andy@yumaworks.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: March 18, 2018

Z. Shelby
ARM
Z. Vial
Schneider-Electric
M. Koster
SmartThings
C. Groves
J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University of Technology

September 14, 2017

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-04

Abstract

For CoAP [RFC7252] Dynamic linking of state updates between resources, either on an endpoint or between endpoints, is defined with the concept of Link Bindings. This specification defines conditional observation attributes that work with Link Bindings or with CoAP Observe [RFC7641].

Editor's note:

o The git repository for the draft is found at https://github.com/core-wg/dynlink

Table of Contents

1. Introduction 3
2. Terminology 3
3. Link Bindings 3
3.1. Binding Methods 4
3.1.1. Polling 5
3.1.2. Observe 5
3.1.3. Push 5
3.2. Link Relation 6
3.3. Binding Attributes 6
3.3.1. Bind Method (bind) 6
3.3.2. Minimum Period (pmin) 6
3.3.3. Maximum Period (pmax) 7
3.3.4. Change Step (st) 7
3.3.5. Greater Than (gt) 7
3.3.6. Less Than (lt) 7
3.3.7. Attribute Interactions 8
4. Binding Table 8
4.1. Binding Interface Description 8
4.2. Resource Observation Attributes 9
5. Security Considerations 10
6. IANA Considerations 11
6.1. Interface Description 11
6.2. Link Relation Type 11
7. Acknowledgements 11
8. Changelog 12
9. References 13
9.1. Normative References 13
9.2. Informative References 13
Appendix A. Examples 13
A.1. Greater Than (gt) example 14
A.2. Greater Than (gt) and Period Max (pmax) example 14

Authors' Addresses 15

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format is a standard for doing Web Linking [RFC5988] in constrained environments.

This specification introduces the concept of a Link Binding, which defines a new link relation type to create a dynamic link between resources over which to exchange state updates. Specifically, a Link Binding is a link for binding the state of 2 resources together such that updates to one are sent over the link to the other. CoRE Link Format representations are used to configure, inspect, and maintain Link Bindings. This specification additionally defines a set of conditional Observe Attributes for use with Link Bindings and with the standalone CoRE Observe [RFC7641] method.

2. Terminology

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

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

Link Binding: A unidirectional logical link between a source resource and a destination resource, over which state information is synchronized.

State Synchronization: Depending on the binding method (Polling, Observe, Push) different REST methods may be used to synchronize the resource values between a source and a destination. The process of using a REST method to achieve this is defined as "State Synchronization". The endpoint triggering the state synchronization is the synchronization initiator.

3. Link Bindings

In a M2M RESTful environment, endpoints may directly exchange the content of their resources to operate the distributed system. For example, a light switch may supply on-off control information that may be sent directly to a light resource for on-off control.

Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human intervention and a so-called commissioning tool. In this specification the abstract relationship between two resources is called a link Binding. The configuration phase necessitates the exchange of binding information so a format recognized by all CoRE endpoints is essential. This specification 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.

3.1. Binding Methods

A binding method defines the rules to generate the web-transfer exchanges that synchronize state between source and destination resources. By using REST methods content is sent from the source resource to the destination resource.

The following table gives a summary of the binding methods defined in this specification.

Name	Identifier	Location	Method
Polling	poll	Destination	GET
Observe	obs	Destination	GET + Observe
Push	push	Source	PUT

Table 1: Binding Method Summary

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.

The binding methods are described in more detail below.

3.1.1. 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 based on the Change Step, Less Than, Greater Than attributes).

3.1.2. Observe

The Observe method creates an observation relationship between the destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the CoAP Observation mechanism [RFC7641] hence this method is only permitted when the resources are made available over CoAP. The binding entry for this method MUST be stored on the destination endpoint. The binding conditions are mapped as query string parameters (see Section 4.2).

3.1.3. 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.

3.2. Link Relation

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format are a natural way to represent binding information. This involves the creation of a new relation type, 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.

3.3. Binding Attributes

Web link attributes allow a fine-grained control of the type of state synchronization along with the conditions that trigger an update. This specification defines the attributes below:

Attribute	Parameter	Value
Binding method	bind	xsd:string
Minimum Period (s)	pmin	xsd:integer (>0)
Maximum Period (s)	pmax	xsd:integer (>0)
Change Step	st	xsd:decimal (>0)
Greater Than	gt	xsd:decimal
Less Than	lt	xsd:decimal

Table 2: Binding Attributes Summary

3.3.1. Bind Method (bind)

This is the identifier of a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

3.3.2. Minimum Period (pmin)

When present, the minimum period indicates the minimum time to wait (in seconds) before triggering a new state synchronization (even if it has changed). In the absence of this parameter, the minimum period is up to the synchronization initiator. The minimum period MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.3.3. Maximum Period (pmax)

When present, the maximum period indicates the maximum time in seconds between two consecutive state synchronizations (regardless if it has changed). In the absence of this parameter, the maximum period is up to the synchronization initiator. The maximum period MUST be greater than zero and MUST be greater than the minimum period parameter (if present) otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.3.4. Change Step (st)

When present, the change step indicates how much the value of a resource SHOULD change before triggering a new state synchronization (compared to the value of the previous synchronization). Upon reception of a query including the st attribute the current value (CurrVal) of the resource is set as the initial value (STinit). Once the resource value differs from the STinit value (i.e. CurrVal >= STinit + ST or CurrVal <= STint - ST) then a new state synchronization occurs. STinit is then set to the state synchronization value and new state synchronizations are based on a change step against this value. The change step MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

Note: Due to the state synchronization based update of STint it may result in that resource value received in two sequential state synchronizations differs by more than st.

3.3.5. Greater Than (gt)

When present, Greater Than indicates the upper limit value the resource value SHOULD cross before triggering a new state synchronization. State synchronization only occurs when the resource value exceeds the specified upper limit value. The actual resource value is used for the synchronization rather than the gt value. If the value continues to rise, no new state synchronizations are generated as a result of gt. If the value drops below the upper limit value and then exceeds the upper limit then a new state synchronization is generated.

3.3.6. Less Than (lt)

When present, Less Than indicates the lower limit value the resource value SHOULD cross before triggering a new state synchronization. State synchronization only occurs when the resource value is less than the specified lower limit value. The actual resource value is used for the synchronization rather than the lt value. If the value

continues to fall no new state synchronizations are generated as a result of lt. If the value rises above the lower limit value and then drops below the lower limit then a new state synchronization is generated.

3.3.7. Attribute Interactions

Pmin, pmax, st, gt and lt may be present in the same query.

If pmin and pmax are present in a query then they take precedence over the other parameters. Thus even if st, gt or lt are met, if pmin has not been exceeded then no state synchronization occurs. Likewise if st, gt or lt have not been met and pmax time has expired then state synchronization occurs. The current value of the resource is used for the synchronization. If pmin time is exceeded and st, gt or lt are met then the current value of the resource is synchronized. If st is also included, a state synchronization resulting from pmin or pmax updates STinit with the synchronized value.

If gt and lt are included gt MUST be greater than lt otherwise an error CoAP error code 4.00 "Bad Request" (or equivalent) MUST be returned.

If st is included in a query with a gt or lt attribute then state synchronizations occur only when the conditions described by st AND gt or st AND gl are met.

4. 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 below. A profile SHOULD allow only one resource table per endpoint.

4.1. Binding Interface Description

This section defines a REST interface for Binding table resources. The interface supports the link-format type.

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 this interface description is intended to be used with the CoRE Link Format, it is applicable for use in any REST interface definition.

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

Interface	if=	Methods	Content-Formats
Binding	core.bnd	GET, POST, DELETE	link-format

Table 3: Binding Interface Description

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

The following example shows requests for adding, retrieving and deleting bindings in a binding table.

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

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

Req: DELETE /bnd/a/light
Res: 2.04 Changed

Req: DELETE /bnd/
Res: 2.04 Changed
```

Figure 1: Binding Interface Example

4.2. Resource Observation Attributes

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [RFC7641] MAY be used to observe any changes in a resource, and receive asynchronous notifications as a result. In addition, a set of query string parameters are defined here to allow a client to control how often a client is interested in receiving notifications and how much a

resource value should change for the new representation to be interesting. These query parameters are described in the following table. A resource using an interface description defined in this specification and marked as Observable in its link description SHOULD support these observation parameters. The Change Step parameter can only be supported on resources with an atomic numeric value.

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

Resource	Parameter	Data Format
Minimum Period	/{resource}?pmin	xsd:integer (>0)
Maximum Period	/{resource}?pmax	xsd:integer (>0)
Change Step	/{resource}?st	xsd:decimal (>0)
Less Than	/{resource}?lt	xsd:decimal
Greater Than	/{resource}?gt	xsd:decimal

Table 4: Resource Observation Attribute Summary

inimum Period: As per Section 3.3.2

aximum Period: As per Section 3.3.3

Change Step: As per Section 3.3.4

Greater Than: As per Section 3.3.5

Less Than: As per Section 3.3.6

5. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this specification. 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.

6. IANA Considerations

6.1. Interface Description

The specification registers the "binding" CoRE interface description link target attribute value as per [RFC6690].

Attribute Value: core.binding

Description: The binding interface is used to manipulate a binding table which describes the link bindings between source and destination resources for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

6.2. Link Relation Type

This specification registers the new "boundto" link relation type as per [RFC5988].

Relation Name: boundto

Description: The purpose of a boundto relation type is to indicate that there is a binding between a source resource and a destination resource for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

Application Data: None

7. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface 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 specification.

8. Changelog

draft-ietf-core-dynlink-03

- General: Reverted to using "gt" and "lt" from "gth" and "lth" for this draft owing to concerns raised that the attributes are already used in LwM2M with the original names "gt" and "lt".
- New author and editor added.

draft-ietf-core-dynlink-02

- General: Changed the name of the greater than attribute "gt" to "gth" and the name of the less than attribute "lt" to "lth" due to conlict with the core resource directory draft lifetime "lt" attribute.
- Clause 6.1: Addressed the editor's note by changing the link target attribute to "core.binding".
- Added Appendix A for examples.

draft-ietf-core-dynlink-01

- General: The term state synchronization has been introduced to describe the process of synchronization between destination and source resources.
- General: The document has been restructured the make the information flow better.
- Clause 3.1: The descriptions of the binding attributes have been updated to clarify their usage.
- Clause 3.1: A new clause has been added to discuss the interactions between the resources.
- Clause 3.4: Has been simplified to refer to the descriptions in 3.1. As the text was largely duplicated.
- Clause 4.1: Added a clarification that individual resources may be removed from the binding table.
- Clause 6: Formailised the IANA considerations.

draft-ietf-core-dynlink Initial Version 00:

- This is a copy of draft-groves-core-dynlink-00

draft-groves-core-dynlink Draft Initial Version 00:

- This initial version is based on the text regarding the dynamic linking functionality in I.D.ietf-core-interfaces-05.
- The WADL description has been dropped in favour of a thorough textual description of the REST API.

9. References

9.1. Normative References

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

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

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

9.2. Informative References

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

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

Appendix A. Examples

This appendix provides some examples of the use of binding attribute / observe attributes.

Note: For brevity the only the method or response code is shown in the header field.

A.1. Greater Than (gt) example

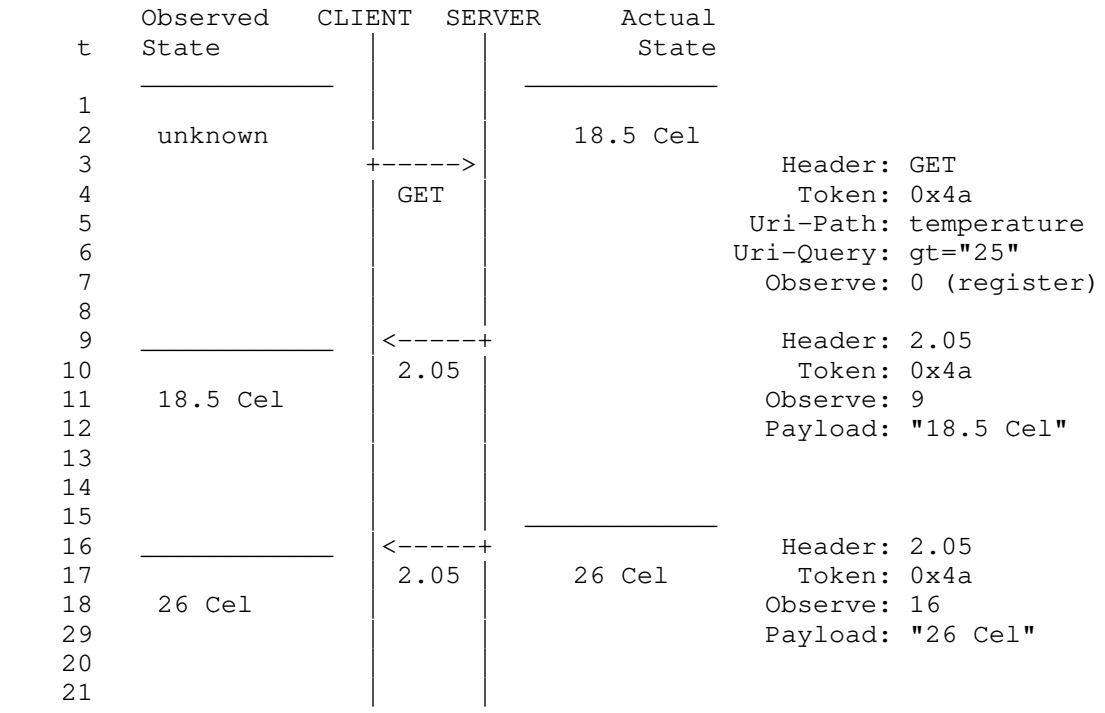
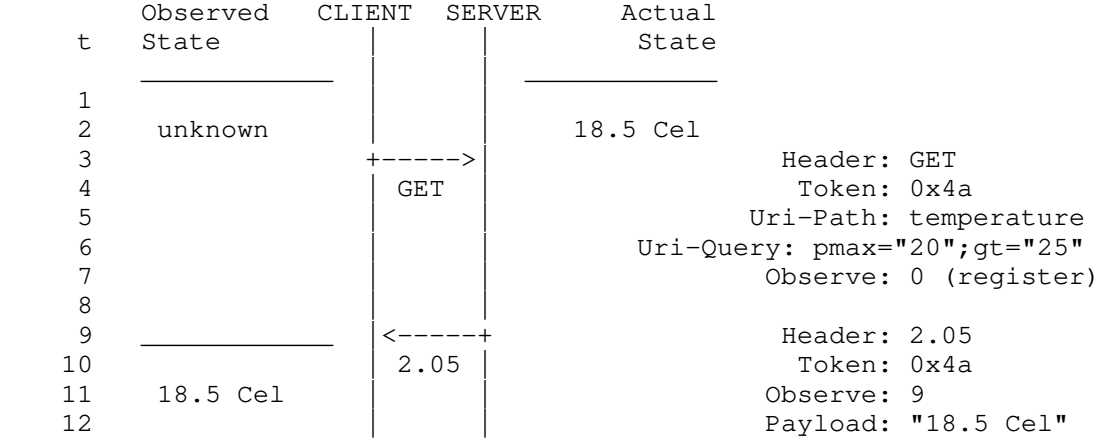


Figure 2: Client Registers and Receives one Notification of the Current State and One of a New State when it passes through the greather than threshold of 25.

A.2. Greater Than (gt) and Period Max (pmax) example



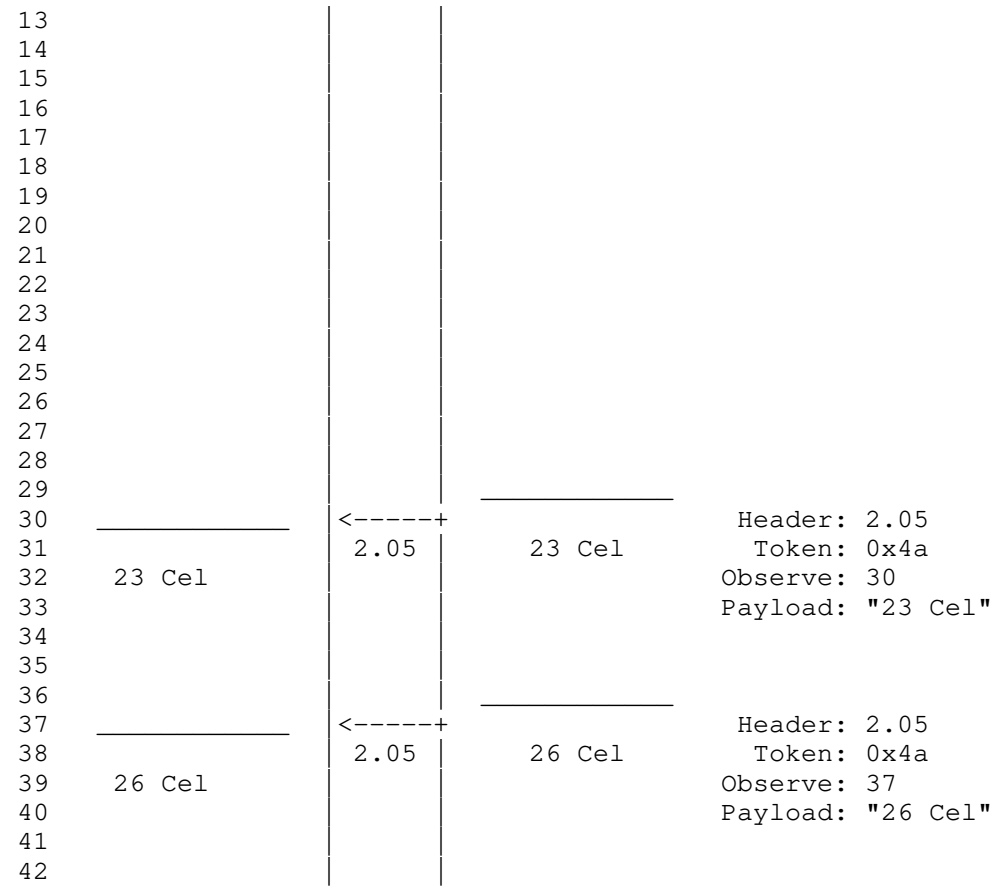


Figure 3: Client Registers and Receives one Notification of the Current State, one when pmax time expires and one of a new State when it passes through the greather than threshold of 25.

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
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smartthings.com

Christian Groves
Huawei
Australia

Email: cngroves.std@gmail.com

Julian Zhu
Huawei
No.127 Jinye Road, Huawei Base, High-Tech Development District
Xi'an, Shaanxi Province
China

Email: jintao.zhu@huawei.com

Bilhanan Silverajan (editor)
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

CoRE Working Group
Internet-Draft
Updates: 7252, 7959 (if approved)
Intended status: Standards Track
Expires: September 6, 2018

C. Amsuess
J. Mattsson
G. Selander
Ericsson AB

March 05, 2018

Echo and Request-Tag
draft-ietf-core-echo-request-tag-01

Abstract

This document specifies several security enhancements to the Constrained Application Protocol (CoAP). Two optional extensions are defined: the Echo option and the Request-Tag option. Each of these options provide additional features to CoAP and protects against certain attacks. The document also updates the processing requirements on the Block options and the Token. The updated Token processing ensures secure binding of responses to requests.

Table of Contents

1. Introduction 2
1.1. Request Freshness 3
1.2. Fragmented Message Body Integrity 3
1.3. Request-Response Binding 4
1.4. Terminology 5
2. The Echo Option 5
2.1. Option Format 5
2.2. Echo Processing 6
2.3. Applications 8
3. The Request-Tag Option 9
3.1. Option Format 9
3.2. Request-Tag Processing 10
3.3. Applications 11
3.3.1. Body Integrity Based on Payload Integrity 11
3.3.2. Multiple Concurrent Blockwise Operations 12
3.4. Rationale for the option properties 13
4. Block2 / ETag Processing 14
5. Token Processing 14
6. IANA Considerations 14
7. Security Considerations 14
8. References 15
8.1. Normative References 15
8.2. Informative References 16
Appendix A. Methods for Generating Echo Option Values 17
Appendix B. Request-Tag Message Size Impact 18
Appendix C. Change Log 18
Authors' Addresses 19

1. Introduction

The initial Constrained Application Protocol (CoAP) suite of specifications ([RFC7252], [RFC7641], and [RFC7959]) was designed with the assumption that security could be provided on a separate layer, in particular by using DTLS ([RFC6347]). However, for some use cases, additional functionality or extra processing is needed to support secure CoAP operations. This document specifies several security enhancements to the Constrained Application Protocol (CoAP).

This document specifies two server-oriented CoAP options, the Echo option and the Request-Tag option, mainly addressing the security features request freshness and fragmented message body integrity,

respectively. The Echo option enables a CoAP server to verify the freshness of a request, verify the aliveness of a client, synchronize state, or force a client to demonstrate reachability at its apparent network address. The Request-Tag option allows the CoAP server to match message fragments belonging to the same request, fragmented using the CoAP Block-Wise Transfer mechanism, which mitigates attacks and enables concurrent blockwise operations. These options in themselves do not replace the need for a security protocol; they specify the format and processing of data which, when integrity protected using e.g. DTLS ([RFC6347]), TLS ([RFC5246]), or OSCORE ([I-D.ietf-core-object-security]), provide the additional security features.

The document also updates the processing requirements on the Block1 option, the Block2 option, and the Token. The updated blockwise processing secure blockwise operations with multiple representations of a particular resource. The updated Token processing ensures secure binding of responses to requests.

1.1. Request Freshness

A CoAP server receiving a request is in general not able to verify when the request was sent by the CoAP client. This remains true even if the request was protected with a security protocol, such as DTLS. This makes CoAP requests vulnerable to certain delay attacks which are particularly incriminating in the case of actuators ([I-D.mattsson-core-coap-actuators]). Some attacks are possible to mitigate by establishing fresh session keys (e.g. performing the DTLS handshake) for each actuation, but in general this is not a solution suitable for constrained environments.

A straightforward mitigation of potential delayed requests is that the CoAP server rejects a request the first time it appears and asks the CoAP client to prove that it intended to make the request at this point in time. The Echo option, defined in this document, specifies such a mechanism which thereby enables the CoAP server to verify the freshness of a request. This mechanism is not only important in the case of actuators, or other use cases where the CoAP operations require freshness of requests, but also in general for synchronizing state between CoAP client and server and to verify aliveness of the client.

1.2. Fragmented Message Body Integrity

CoAP was designed to work over unreliable transports, such as UDP, and include a lightweight reliability feature to handle messages which are lost or arrive out of order. In order for a security protocol to support CoAP operations over unreliable transports, it

must allow out-of-order delivery of messages using e.g. a sliding replay window such as described in Section 4.1.2.6 of DTLS ([RFC6347]).

The Block-Wise Transfer mechanism [RFC7959] extends CoAP by defining the transfer of a large resource representation (CoAP message body) as a sequence of blocks (CoAP message payloads). The mechanism uses a pair of CoAP options, Block1 and Block2, pertaining to the request and response payload, respectively. The blockwise functionality does not support the detection of interchanged blocks between different message bodies to the same resource having the same block number. This remains true even when CoAP is used together with a security protocol such as DTLS or OSCORE, within the replay window ([I-D.mattsson-core-coap-actuators]), which is a vulnerability of CoAP when using RFC7959.

A straightforward mitigation of mixing up blocks from different messages is to use unique identifiers for different message bodies, which would provide equivalent protection to the case where the complete body fits into a single payload. The ETag option [RFC7252], set by the CoAP server, identifies a response body fragmented using the Block2 option. This document defines the Request-Tag option for identifying the request body fragmented using the Block1 option, similar to ETag, but ephemeral and set by the CoAP client.

1.3. Request-Response Binding

A fundamental requirement of secure REST operations is that the client can bind a response to a particular request. In HTTPS this is assured by the ordered and reliable delivery as well as mandating that the server sends responses in the same order that the requests were received.

The same is not true for CoAP where the server can return responses in any order. Concurrent requests are instead differentiated by their Token. Unfortunately, CoAP [RFC7252] does not treat Token as a cryptographically important value and does not give stricter guidelines than that the tokens currently "in use" SHOULD (not SHALL) be unique. If used with security protocol not providing bindings between requests and responses (e.g. DTLS and TLS) token reuse may result in situations where a client matches a response to the wrong request (see e.g. Section 2.3 of [I-D.mattsson-core-coap-actuators]). Note that mismatches can also happen for other reasons than a malicious attacker, e.g. delayed delivery or a server sending notifications to an uninterested client.

A straightforward mitigation is to mandate clients to never reuse tokens until the traffic keys have been replaced. As there may be

any number of responses to a request (see e.g. [RFC7641]), the easiest way to accomplish this is to implement the token as a counter and never reuse any tokens at all. This document updates the Token processing in [RFC7252] to always assure a cryptographically secure binding of responses to requests.

1.4. 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].

Unless otherwise specified, the terms "client" and "server" refers to "CoAP client" and "CoAP server", respectively, as defined in [RFC7252].

The terms "payload" and "body" of a message are used as in [RFC7959]. The complete interchange of a request and a response body is called a (REST) "operation". An operation fragmented using [RFC7959] is called a "blockwise operation". A blockwise operation which is fragmenting the request body is called a "blockwise request operation". A blockwise operation which is fragmenting the response body is called a "blockwise response operation".

Two blockwise operations between the same endpoint pair on the same resource are said to be "concurrent" if a block of the second request is exchanged even though the client still intends to exchange further blocks in the first operation. (Concurrent blockwise request operations are impossible with the options of [RFC7959] because the second operation's block overwrites any state of the first exchange.).

The Echo and Request-Tag options are defined in this document. The concept of two messages being "Request-Tag-matchable" is defined in Section 3.1.

2. The Echo Option

The Echo option is a server-driven challenge-response mechanism for CoAP. The Echo option value is a challenge from the server to the client included in a CoAP response and echoed in one or more CoAP request.

2.1. Option Format

The Echo Option is elective, safe-to-forward, not part of the cache-key, and not repeatable, see Figure 1.

No.	C	U	N	R	Name	Format	Length	Default	E
TBD			x		Echo	opaque	4-40	(none)	x

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable,
E = Encrypt and Integrity Protect (when using OSCORE)

Figure 1: Echo Option Summary

[Note to RFC editor: If this document is not released together with OSCORE but before it, the following paragraph and the "E" column above need to move into OSCORE.]

The Echo option value is generated by the server, and its content and structure are implementation specific. Different methods for generating Echo option values are outlined in Appendix A. Clients and intermediaries MUST treat an Echo option value as opaque and make no assumptions about its content or structure.

When receiving an Echo option in a request, the server MUST be able to verify that the Echo option value was generated by the server as well as the point in time when the Echo option value was generated.

2.2. Echo Processing

The Echo option MAY be included in any request or response (see Section 2.3 for different applications), but the Echo option MUST NOT be used with empty CoAP requests (i.e. Code=0.00).

If the server receives a request which has freshness requirements, the request does not contain a fresh Echo option value, and the server cannot verify the freshness of the request in some other way, the server MUST NOT process the request further and SHOULD send a 4.01 Unauthorized response with an Echo option.

The application decides under what conditions a CoAP request to a resource is required to be fresh. These conditions can for example include what resource is requested, the request method and other data in the request, and conditions in the environment such as the state of the server or the time of the day.

The server may also include the Echo option in a response to verify the aliveness of a client, to synchronize state, or to force a client to demonstrate reachability at their apparent network address.

Upon receiving a 4.01 Unauthorized response with the Echo option, the client SHOULD resend the original request with the addition of an Echo option with the received Echo option value. The client MAY send a different request compared to the original request. Upon receiving any other response with the Echo option, the client SHOULD echo the Echo option value in a next request to the server. The client MAY include the same Echo option value in several different requests to the server.

Upon receiving a request with the Echo option, the server determines if the request has freshness requirement. If the request does not have freshness requirements, the Echo option MAY be ignored. If the request has freshness requirements and the server cannot verify the freshness of the request in some other way, the server MUST verify that the Echo option value was generated by the server; otherwise the request is not processed further. The server MUST then calculate the round-trip time RTT = (t1 - t0), where t1 is the request receive time and t0 is the transmit time of the response that included the specific Echo option value. The server MUST only accept requests with a round-trip time below a certain threshold T, i.e. RTT < T, otherwise the request is not processed further, and an error message MAY be sent. The threshold T is application specific, its value depends e.g. on the freshness requirements of the request. An example message flow is illustrated in Figure 2.

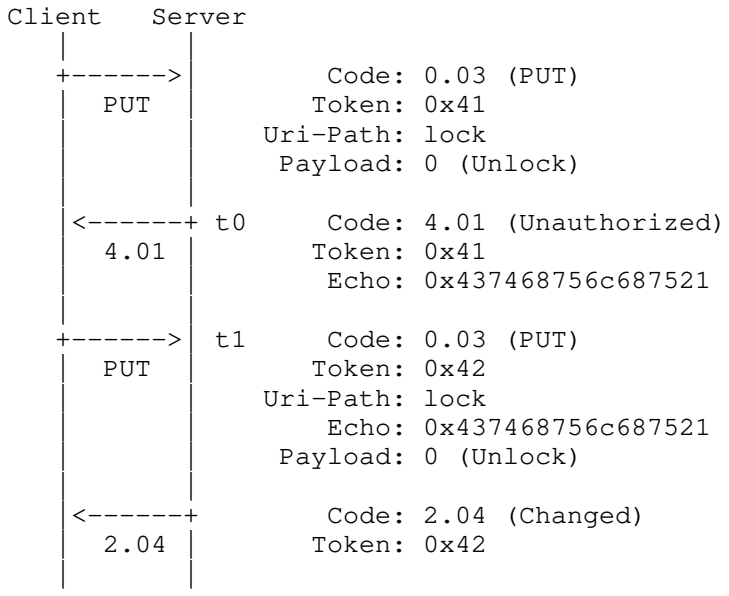


Figure 2: Example Echo Option Message Flow

When used to serve freshness requirements (including client aliveness and state synchronizing), CoAP requests containing the Echo option MUST be integrity protected, e.g. using DTLS, TLS, or OSCORE ([I-D.ietf-core-object-security]). When used to demonstrate reachability at their apparent network address, the Echo option MAY be used without protection.

Note that the server does not have to synchronize the time used for the Echo timestamps with any other party. If the server loses time synchronization, e.g. due to reboot, it MUST reject all Echo values that was created before time synchronization was lost.

CoAP-CoAP proxies MUST relay the Echo option unmodified. The CoAP server side of CoAP-HTTP proxies MAY request freshness, especially if they have reason to assume that access may require it (e.g. because it is a PUT or POST); how this is determined is out of scope for this document. The CoAP client side of HTTP-CoAP-Proxies SHOULD respond to Echo challenges themselves if they know from the recent establishing of the connection that the HTTP request is fresh. Otherwise, they SHOULD respond with 503 Service Unavailable, Retry-After: 0 and terminate any underlying Keep-Alive connection. They MAY also use other mechanisms to establish freshness of the HTTP request that are not specified here.

2.3. Applications

1. Actuation requests often require freshness guarantees to avoid accidental or malicious delayed actuator actions. In general, all non-safe methods (e.g. POST, PUT, DELETE) may require freshness guarantees for secure operation.

2. To avoid additional roundtrips for applications with multiple actuator requests in rapid sequence between the same client and server, the server may use the Echo option (with a new value) in response to a request containing the Echo option. The client then uses the Echo option with the new value in the next actuation request, and the server compares the receive time accordingly.

3. If a server reboots during operation it may need to synchronize state with requesting clients before continuing the interaction. For example, with OSCORE it is possible to reuse a partly persistently stored security context by synchronizing the Partial IV (sequence number) using the Echo option.

4. When a device joins a multicast/broadcast group the device may need to synchronize state or time with the sender to ensure that the received message is fresh. By synchronizing time with the

broadcaster, time can be used for synchronizing subsequent broadcast messages. A server MUST NOT synchronize state or time with clients which are not the authority of the property being synchronized. E.g. if access to a server resource is dependent on time, then the client MUST NOT set the time of the server.

5. A server that sends large responses to unauthenticated peers SHOULD mitigate amplification attacks such as described in Section 11.3 of [RFC7252] (where an attacker would put a victim's address in the source address of a CoAP request). For this purpose, the server MAY ask a client to Echo its request to verify its source address. This needs to be done only once per peer and limits the range of potential victims from the general Internet to endpoints that have been previously in contact with the server. For this application, the Echo option can be used in messages that are not integrity protected, for example during discovery.

6. A server may want to verify the aliveness of a client by responding with an Echo option.

3. The Request-Tag Option

The Request-Tag is intended for use as a short-lived identifier for keeping apart distinct blockwise request operations on one resource from one client. It enables the receiving server to reliably assemble request payloads (blocks) to their message bodies, and, if it chooses to support it, to reliably process simultaneous blockwise request operations on a single resource. The requests must be integrity protected in order to protect against interchange of blocks between different message bodies.

3.1. Option Format

The Request-Tag option is not critical, safe to forward, and part of the cache key as illustrated in Figure 3.

No.	C	U	N	R	Name	Format	Length	Default	E
TBD					Request-Tag	opaque	0-8	(none)	*

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable,
E = Encrypt and Integrity Protect (when using OSCORE)

Figure 3: Request-Tag Option Summary

[Note to RFC editor: If this document is not released together with OSCORE but before it, the following paragraph and the "E" column above need to move into OSCORE.]

Request-Tag, like the block options, is a special class E option in terms of OSCORE processing (see Section 4.3.1.2 of [I-D.ietf-core-object-security]): The Request-Tag MAY be an inner or outer option. The inner option is encrypted and integrity protected between client and server, and provides message body identification in case of end-to-end fragmentation of requests. The outer option is visible to proxies and labels message bodies in case of hop-by-hop fragmentation of requests.

The Request-Tag option is only used in the request messages of blockwise request operations.

Two messages are defined to be Request-Tag-matchable if and only if they are sent from and to the same end points (including security associations), and target the same URI (precisely: target the same endpoint and cache-key except for cache-key options that are related to blockwise), and if either neither carries a Request-Tag option, or both carry exactly one Request-Tag option and the option values are of same length and content.

The Request-Tag mechanism is applied independently on the server and client sides of CoAP-CoAP proxies as are the block options, though given it is safe to forward, a proxy is free to just forward it when processing an operation. CoAP-HTTP proxies and HTTP-CoAP proxies can use Request-Tag on their CoAP sides; it is not applicable to HTTP requests.

For each separate blockwise request operation, the client can choose a Request-Tag value, or choose not to set a Request-Tag. Creating a new request operation whose messages are Request-Tag-matchable to a previous operation is called request tag recycling. Clients MUST NOT recycle a request tag unless the first operation has concluded. What constitutes a concluded operation depends on the application, and is outlined individually in Section 3.3.

Clients are encouraged to generate compact messages. This means sending messages without Request-Tag options whenever possible, and using short values when the absent option can not be recycled.

3.2. Request-Tag Processing

A server MUST NOT act on any two blocks in the same blockwise request operation that are not Request-Tag-matchable. This rule applies independent of whether the request actually carries a Request-Tag

option (if not, the request can only be acted on together with other messages not carrying the option, as per matchability definition).

As not all messages from the same source can be combined any more, a block not matchable to the first Block1 cannot overwrite context kept for an operation under a different tag (cf. [RFC7959] Section 2.5). The server is still under no obligation to keep state of more than one transaction. When an operation is in progress and a second one cannot be served at the same time, the server SHOULD respond to the second request with a 5.03 (Service Unavailable) response code and indicate the time it is willing to wait for additional blocks in the first operation using the Max-Age option, as specified in Section 5.9.3.4 of [RFC7252]. (Alternatively, the server can cancel the original operation, especially if it is already likely to time out. Cancelling it unconditionally is the behavior that could be expected of a Request-Tag unaware server.)

A server receiving a Request-Tag MUST treat it as opaque and make no assumptions about its content or structure.

Two messages being Request-Tag-matchable is a necessary but not sufficient condition for being part of the same operation. They can still be treated as independent messages by the server (e.g. when it sends 2.01/2.04 responses for every block), or initiate a new operation (overwriting kept context) when the later message carries Block1 number 0.

Note that RFC 7959 already implies that the cache key is the element that binds exchanges together to operations (together with the request's source endpoint), but is not explicit about it; therefore, the above rules are spelt out here.

3.3. Applications

3.3.1. Body Integrity Based on Payload Integrity

When a client fragments a request body into multiple message payloads, even if the individual messages are integrity protected, it is still possible for a man-in-the-middle to maliciously replace later operation's blocks with earlier operation's blocks (see Section 2.5 of [I-D.mattsson-core-coap-actuators]). Therefore, the integrity protection of each block does not extend to the operation's request body.

In order to gain that protection, use the Request-Tag mechanism as follows:

- The individual exchanges MUST be integrity protected end-to-end between client and server.

- The client MUST NOT recycle a request tag unless the previous blockwise request operation that used matchable Request-Tags has concluded.

- The client MUST NOT regard a blockwise request operation as concluded unless all of the messages the client previously sent in the operation have been confirmed by the message integrity protection mechanism, or are considered invalid by the server if replayed.

  Typically, in OSCORE, these confirmations can result either from the client receiving an OSCORE response message matching the request (an empty ACK is insufficient), or because the message's sequence number is old enough to be outside the server's receive window.

  In DTLS, this can only be confirmed if the request message was not retransmitted, and was responded to.

Authors of other documents (e.g. [I-D.ietf-core-object-security]) are invited to mandate this behavior for clients that execute blockwise interactions over secured transports. In this way, the server can rely on a conforming client to set the Request-Tag option when required, and thereby conclude on the integrity of the assembled body.

Note that this mechanism is implicitly implemented when the security layer guarantees ordered delivery (e.g. CoAP over TLS [RFC8323]). This is because with each message, any earlier operation can be regarded as concluded by the client, so it never needs to set the Request-Tag option unless it wants to perform concurrent operations.

3.3.2. Multiple Concurrent Blockwise Operations

CoAP clients, especially CoAP proxies, may initiate a blockwise request operation to a resource, to which a previous one is already in progress, and which the new request should not cancel. A CoAP proxy would be in such a situation when it forwards operations with the same cache-key options but possibly different payloads.

When a client fragments an initial message as part of a blockwise request operation, it can do so without a Request-Tag option set. For this application, an operation can be regarded as concluded when a final Block1 option has been sent and acknowledged, or when the client chose not to continue with the operation (e.g. by user choice,

or in the case of a proxy when it decides not to take any further messages in the operation due to a timeout). When another concurrent blockwise request operation is made (i.e. before the operation is concluded), the client can not recycle the request tag, and has to pick a new one. The possible outcomes are:

- The server responds with a successful code.

  The second concurrent blockwise operations can then continue.

  The first operation might have been cancelled by that (typical of servers that only support a single blockwise operation), in which case its resumption will result in a 4.08 Request Entity Incomplete error.

- The server responds 5.03 Service Unavailable with a Max-Age option to indicate when it is likely to be available again.

  This can indicate that the server supports Request-Tag, but still is not prepared to handle concurrent requests. The client should wait for as long as the response is valid, and then retry the operation, which may not need to carry a Request-Tag option by then any more.

  In this, the proxy can indicate the anticipated delay by sending a 5.03 Service Unavailable response itself.

Note that a correctly implemented Request-Tag unaware proxy in the same situation would need to make a choice to either send a 5.03 with Max-Age by itself (holding off the second operation), or to commence the second operation and reject any further requests on the first operation with 4.08 Request Entity Incompelte errors by itself without forwarding them.

3.4. Rationale for the option properties

The Request-Tag option used to be critical and unsafe to forward in earlier revisions of this draft.

Given that supporting it will be mandated for where it is used for its security properties, the choice of whether it is mandatory or safe to forward can be made as required for the multiple concurrent operations use case. For those cases, Request-Tag is the proxy-safe elective option suggested in [RFC7959] Section 2.4 last paragraph.

4. Block2 / ETag Processing

The same security properties as in Section 3.3.1 can be obtained for blockwise response operations. The threat model here is not an attacker (because the response is made sure to belong to the current request by the security layer), but blocks in the client's cache.

Analogous rules to Section 3.2 are already in place for assembling a response body in Section 2.4 of [RFC7959].

To gain equivalent protection to Section 3.3.1, a server MUST use the Block2 option in conjunction with the ETag option ([RFC7252], Section 5.10.6), and MUST NOT use the same ETag value for different representations of a resource.

5. Token Processing

This section updates the Token processing in Section 5.3.1 of [RFC7252] by adding the following text:

When CoAP is used with a security protocol not providing bindings between requests and responses, the client MUST NOT reuse tokens until the traffic keys have been replaced. The easiest way to accomplish this is to implement the Token as a counter, this approach SHOULD be followed.

6. IANA Considerations

This document adds the following option numbers to the "CoAP Option Numbers" registry defined by [RFC7252]:

Number	Name	Reference
TBD1	Echo	[RFC XXXX]
TBD2	Request-Tag	[RFC XXXX]

Figure 4: CoAP Option Numbers

7. Security Considerations

Servers SHOULD NOT put any privacy sensitive information in the Echo or Request-Tag option values. Unencrypted timestamps MAY reveal information about the server such as its wall clock time or location. Servers MUST use a monotonic clock to generate timestamps and compute round-trip times. Servers SHOULD NOT use wall clock time for

timestamps, as wall clock time is not monotonic, may reveal that the server will accept expired certificates, or reveal the server's location. Use of non-monotonic clocks is not secure as the server will accept expired Echo option values if the clock is moved backward. The server will also reject fresh Echo option values if the clock is moved forward. An attacker may be able to affect the server's wall clock time in various ways such as setting up a fake NTP server or broadcasting false time signals to radio-controlled clocks. Servers SHOULD use the time since reboot measured in some unit of time. Servers MAY reset the timer periodically even when not rebooting.

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of the Echo option. If no true random number generator is available, a truly random seed must be provided from an external source.

An Echo value with 64 (pseudo-)random bits gives the same theoretical security level against forgeries as a 64-bit MAC (as used in e.g. AES_128_CCM_8). In practice, forgery of an Echo option value is much harder as an attacker must also forge the MAC in the security protocol. The Echo option value MUST contain 32 (pseudo-)random bits that are not predictable for any other party than the server, and SHOULD contain 64 (pseudo-)random bits. A server MAY use different security levels for different uses cases (client aliveness, request freshness, state synchronization, network address reachability, etc.).

The security provided by the Echo and Request-Tag options depends on the security protocol used. CoAP and HTTP proxies require (D)TLS to be terminated at the proxies. The proxies are therefore able to manipulate, inject, delete, or reorder options or packets. The security claims in such architectures only hold under the assumption that all intermediaries are fully trusted and have not been compromised.

Servers that use the List of Cached Random Values and Timestamps method described in Appendix A may be vulnerable to resource exhaustion attacks. On way to minimizing state is to use the Integrity Protected Timestamp method described in Appendix A.

8. References

8.1. Normative References

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

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

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

8.2. Informative References

[I-D.ietf-core-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-09 (work in progress), March 2018.

[I-D.mattsson-core-coap-actuators] Mattsson, J., Fornehed, J., Selander, G., Palombini, F., and C. Amsuess, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-04 (work in progress), March 2018.

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

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

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

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

Appendix A. Methods for Generating Echo Option Values

The content and structure of the Echo option value are implementation specific and determined by the server. Use of one of the mechanisms outlined in this section is RECOMMENDED.

Different mechanisms have different tradeoffs between the size of the Echo option value, the amount of server state, the amount of computation, and the security properties offered.

- Integrity Protected Timestamp. One method is to construct the Echo option value as an integrity protected timestamp. The timestamp can have different resolution and range. A 32-bit timestamp can e.g. give a resolution of 1 second with a range of 136 years. The (pseudo-)random secret key is generated by the server and not shared with any other party. The use of truncated HMAC-SHA-256 is RECOMMENDED. With a 32-bit timestamp and a 64-bit MAC, the size of the Echo option value is 12 bytes and the Server state is small and constant. If the server loses time synchronization, e.g. due to reboot, the old key MUST be deleted and replaced by a new random secret key. A server MAY also want to encrypt its timestamps, depending on the choice of encryption algorithms, this may require a nonce to be included in the Echo option value.

  ```
  Echo option value: timestamp t0, MAC(k, t0)
  Server State: secret key k
  ```

- List of Cached Random Values and Timestamps. An alternative method is to construct the Echo option value as a (pseudo-)random byte string. The server caches a list containing the random byte strings and their transmission times. Assuming 64-bit random values and 32-bit timestamps, the size of the Echo option value is 8 bytes and the amount of server state is 12n bytes, where n is the number of active Echo Option values. If the server loses time synchronization, e.g. due to reboot, the entries in the old list MUST be deleted.

  ```
  Echo option value: random value r
  Server State: random value r, timestamp t0
  ```

A server MAY use different methods and security levels for different uses cases (client aliveness, request freshness, state synchronization, network address reachability, etc.).

Appendix B. Request-Tag Message Size Impact

In absence of concurrent operations, the Request-Tag mechanism for body integrity (Section 3.3.1) incurs no overhead if no messages are lost (more precisely: in OSCORE, if no operations are aborted due to repeated transmission failure; in DTLS, if no packages are lost), or when blockwise request operations happen rarely (in OSCORE, if only one request operation with losses within the replay window).

In those situations, no message has any Request-Tag option set, and that can be recycled indefinitely.

When the absence of a Request-Tag option can not be recycled any more within a security context, the messages with a present but empty Request-Tag option can be used (1 Byte overhead), and when that is used-up, 256 values from one byte long options (2 Bytes overhead) are available.

In situations where those overheads are unacceptable (e.g. because the payloads are known to be at a fragmentation threshold), the absent Request-Tag value can be made usable again:

- In DTLS, a new session can be established.
- In OSCORE, the sequence number can be artificially increased so that all lost messages are outside of the replay window by the time the first request of the new operation gets processed, and all earlier operations can therefore be regarded as concluded.

Appendix C. Change Log

[The editor is asked to remove this section before publication.]

- Major changes since draft-ietf-core-echo-request-tag-00:
  - Reworded the Echo section.
  - Added rules for Token processing.
  - Added security considerations.
  - Added actual IANA section.
  - Made Request-Tag optional and safe-to-forward, relying on blockwise to treat it as part of the cache-key

  * Dropped use case about OSCORE outer-blockwise (the case went away when its Partial IV was moved into the Object-Security option)

- Major changes since draft-amsuess-core-repeat-request-tag-00:

  * The option used for establishing freshness was renamed from "Repeat" to "Echo" to reduce confusion about repeatable options.

  * The response code that goes with Echo was changed from 4.03 to 4.01 because the client needs to provide better credentials.

  * The interaction between the new option and (cross) proxies is now covered.

  * Two messages being "Request-Tag matchable" was introduced to replace the older concept of having a request tag value with its slightly awkward equivalence definition.

Authors' Addresses

Christian Amsuess

Email: christian@amsuess.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

Z. Shelby
ARM
M. Vial
Schneider-Electric
M. Koster
SmartThings
C. Groves

J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University of Technology
March 05, 2018

Reusable Interface Definitions for Constrained RESTful Environments
draft-ietf-core-interfaces-11

Abstract

This document defines a set of Constrained RESTful Environments (CoRE) Link Format Interface Descriptions [RFC6690] applicable for use in constrained environments. These include the: Actuator, Parameter, Read-only parameter, Sensor, Batch, Linked Batch and Link List interfaces.

The Batch, Linked Batch and Link List interfaces make use of resource collections. This document further describes how collections relate to interfaces.

Many applications require a set of interface descriptions in order provide the required functionality. This document defines an Interface Description atribute value to describe resources conforming to a particular interface.

Editor's notes:

- The git repository for the draft is found at https://github.com/core-wg/interfaces

Status of This Memo

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

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

working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on September 6, 2018.

Table of Contents

1. Introduction 3
2. Terminology 4
3. Collections 5
  3.1. Introduction to Collections 5
  3.2. Use Cases for Collections 5
  3.3. Content-Formats for Collections 6
  3.4. Link Embedding 6
  3.5. Links and Items in Collections 7
  3.6. Queries on Collections 8
  3.7. Observing Collections 8
  3.8. Collection Types 8
4. Interface Descriptions 9
  4.1. Link List 11
  4.2. Batch 11
  4.3. Linked Batch 12
  4.4. Sensor 13
  4.5. Parameter 14
  4.6. Read-only Parameter 14
  4.7. Actuator 14
5. Security Considerations 15
6. IANA Considerations 15

6.1. Link List 15
6.2. Batch 16
6.3. Linked Batch 16
6.4. Sensor 16
6.5. Parameter 16
6.6. Read-only parameter 17
6.7. Actuator 17
7. Acknowledgements 17
8. Changelog 18
9. References 21
9.1. Normative References 21
9.2. Informative References 21
Appendix A. Current Usage of Interfaces and Function Sets . . . 23
A.1. Constrained RESTful Environments (CoRE) Link Format (IETF) 23
A.2. CoRE Resource Directory (IETF) 23
A.3. Open Connectivity Foundation (OCF) 23
A.4. oneM2M 24
A.5. OMA LWM2M 25
Appendix B. Resource Profile example 25
Authors' Addresses 26

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format is a standard for doing Web Linking [RFC8288] in constrained environments. SenML [I-D.ietf-core-senml] is a simple data model and representation format for composite and complex structured resources. CoRE Link-Format and SenML can be used by CoAP [RFC7252] or HTTP servers.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop. Machine application clients must be able to adapt to different resource organizations without advance knowledge of the specific data structures hosted by each connected thing. The use of Web Linking for the description and discovery of resources hosted by constrained origin servers is specified by CoRE Link Format [RFC6690]. CoRE Link Format additionally defines a link attribute for interface description ("if") that can be used to describe the REST interface of a resource, and may include a link to a description document.

This document defines a set of Link Format interface descriptions for some common design patterns that enable the server side composition and organization, and client side discovery and consumption, of

machine resources using Web Linking. A client discovering the "if" link attribute will be able to consume resources based on its knowledge of the expected interface types. In this sense the Interface Type acts in a similar way as a Content-Format, but as a selector for a high level functional abstraction.

An interface description describes a resource in terms of its associated content formats, data types, URI templates, REST methods, parameters, and responses. Basic interface descriptions are defined for sensors, and actuators.

A set of collection types is defined for organizing resources for discovery, and for various forms of bulk interaction with resource sets using typed embedding links.

This document first defines the concept of collection interface descriptions. It then defines a number of generic interface descriptions that may be used in contrained environments. Several of these interface descriptions utilise collections.

Whilst this document assumes the use of CoAP [RFC7252], the REST interfaces described can also be realized using HTTP [RFC7230].

2. Terminology

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

This document requires readers to be familiar with all the terms and concepts that are discussed in [RFC8288] and [RFC6690]. This document makes use of the following additional terminology:

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

Interface Description: The Interface Description describes the generic REST interface to interact with a resource or a set of resources. Its use is described via the Interface Description 'if' attribute which is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource. One can think of this as describing verbs usable on a resource.

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

3. Collections

3.1. Introduction to Collections

A Collection is a resource which represents one or more related resources. [RFC6573] describes the "item" and "collection" Link Relation. An "item" link relation identifies a member of collection. A "collection" indicates the collection that an item is a member of. For example, a collection might be a resource representing a catalog of products, while an item is a resource related to an individual product.

Section 1.2.2/[RFC6690] also describes resource collections.

This document uses the concept of "collection" and applies it to interface descriptions. A collection interface description consists of a set of links and a set of items pointed to by the links which may be sub-resources of the collection resource. The collection interface descriptions described in this document are Link List, Batch and Linked Batch.

The links in a collection are represented in CoRE Link-Format Content-Formats including JSON and CBOR variants, and the items in the collection may be represented by senml, including JSON and CBOR variants. In general, a collection may support items of any available Content-Format.

A particular resource item may be a member of more than one collection at a time by being linked to, but may only be a subresource of one collection.

Some collections may have pre-configured items and links, and some collections may support dynamic creation and removal of items and links. Likewise, modification of items in some collections may be permitted, and not in others.

Links in collections may be selected for processing by a particular request by using Query Filtering as described in CoRE Link-Format [RFC6690].

3.2. Use Cases for Collections

Collections may be used to provide gradual reveal of resources on an endpoint. There may be a small set of links at the .well-known/core location, which may in turn point to other collections of resources that represent device information, device configuration, device management, and various functional clusters of resources on the device.

A collection may be used to group a set of like resources for bulk state update or actuation. For example, the brightness control resources of a number of luminaries may be grouped by linking to them in a collection. The collection type may support receiving a single update from a client and sending that update to each resource item in the collection.

Items may be sub-resources of the collection resource. This enables updates to multiple items in the collection to be processed together within the context of the collection resource.

3.3. Content-Formats for Collections

The collection interfaces by default use CoRE Link-Format for the link representations and SenML or text/plain for representations of items. The examples given are for collections that expose resources and links in these formats. In addition, a new "collection" Content-Format is defined based on the SenML framework which represents both links and items in the collection.

The choice of whether to return a representation of the links or of the items or of the collection format is determined by the Accept header option in the request. Likewise, the choice of updating link metadata or item data or the collection resource itself is determined by the Content-Format option in the header of the update request operation.

The default Content-Formats for collection types described in this document are:

Links: application/link-format, application/link-format+json

Items: application/senml+json, text/plain

3.4. Link Embedding

Collections may provide resource encapsulation by supporting link embedding. Link embedding may be used to provide a single resource with which a client may interact to obtain a set of related resource values. This is analogous to an image tag (link) causing the image to display inline in a browser window. Link embedding enables the bulk processing of items in the collection using a single operation targeting the collection resource. Performing a GET on a collection resource may return a single representation containing all of the embedded linked resources. For example, a collection for manufacturer parameters may consist of manufacturer name, date of manufacture, location of manufacture, and serial number resources which can be read as a single senml data object.

A subset of resources in the collection may be selected for operation using Query Filtering. Bulk Read operations using GET return a SenML representation of all selected resources. Bulk item Update operations using PUT or POST apply the payload document to all selected resource items in the collection, using either a Batch or Group update policy. A Batch update is performed by applying the resource values in the payload document to all resources in the collection that match any resource name in the payload document. Group updates are performed by applying the payload document to each item in the collection. Group updates are indicated by the link relation type rel="grp" in the link.

3.5. Links and Items in Collections

Links use CoRE Link-Format representation by default and may point to any resource reachable from the context of the collection. This includes links to resources with absolute paths as well as links that point to other network locations, if the context of the collection allows. Links to sub-resources in the collection MUST have a path-element starting with the resource name, as per [RFC3986]. Links to resources in the global context MUST start with a root path identifier [RFC8288]. Links to other collections are formed per [RFC3986].

Examples of links:

</sen/>;if="core.lb": Link to the /sen/ collection describing it as a core.lb type collection (Linked Batch)

</sen/>;rel="grp": Link to the /sen/ collection indicating that /sen/ is a member of a group in the collection in which the link appears.

</sen/temp>;rt="temperature": A link to the temp resource with an absolute path.

<temp>;rt="temperature": Link to the temp subresource of the collection in which this link appears.

<temp>;anchor="/sen/": A link to the temp subresource of the collection /sen/ which is assumed not to be a subresource of the collection in which the link appears, but is expected to be identified in the collection by resource name.

Links in the collection MAY be Read, Updated, Added, or Removed using the CoRE Link-Format or JSON Merge-Patch Content-Formats on the collection resource. Reading links uses the GET method and returns an array or list containing the link-values of all selected links.

Links may be added to the collection using POST or PATCH methods. Updates to links MUST use the PATCH method and MAY use query filtering to select links for updating. The PATCH method on links MUST use the JSON Merge-Patch Content-Format (application/merge-patch+json) specified in [RFC7396].

Items in the collection SHOULD be represented using the SenML (application/senml+json) or plain text (text/plain) Content-Formats, depending on whether the representation is of a single data point or multiple data points. Items MAY be represented using any supported Content-Format.

3.6. Queries on Collections

Collections MAY support query filtering as defined in CoRE Link-Format [RFC6690]. Operations targeting either the links or the items MAY select a subset of links and items in the collection by using query filtering. The Content-Format specified in the request header selects whether links or items are targeted by the operation.

3.7. Observing Collections

Resource Observation via [I-D.ietf-core-dynlink] using CoAP [RFC7252] MAY be supported on items in a collection. A subset of the conditional observe parameters MAY be specified to apply. In most cases pmin and pmax are useful. Resource observation on a collection's resource returns the collection representation. Observation Responses, or notifications, SHOULD provide the collection representations in SenML Content-Format. Notifications MAY include multiple observations of the collection resource, with SenML time stamps indicating the observation times.

3.8. Collection Types

There are three collection types defined in this document:

Collection Type	if=	Content-Format
Link List	core.ll	link-format
Batch	core.b	link-format, senml
Linked Batch	core.lb	link-format, senml

Table 1: Collection Type Summary

The interface description defined in this document offer a deeper explanation of the methods and functions that may be applied to the three collections.

4. Interface Descriptions

This section defines REST interfaces for Sensor, Parameter, Read-Only Paramter and Actuator resource types, in addition to the Link List, Batch and Linked Batch collection types. Each type is described along with its Interface Description attribute value, valid methods and content formats. These are shown for each interface in the table below.

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

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

Interface	if=	Methods	Content-Formats
Link List	core.ll	GET	link-format
Batch	core.b	GET, PUT, POST	link-format, senml
Linked Batch	core.lb	GET, PUT, POST, DELETE	link-format, senml
Sensor	core.s	GET	link-format, text/plain
Parameter	core.p	GET, PUT	link-format, text/plain
Read-only Parameter	core.rp	GET	link-format, text/plain
Actuator	core.a	GET, PUT, POST	link-format, text/plain

Table 2: Interface Description Summary

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

```
Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s/>;rt="simple.sen";if="core.b",
</s/light>;rt="simple.sen.lt";if="core.s",
</s/temp>;rt="simple.sen.tmp";if="core.s";obs,
</s/humidity>;rt="simple.sen.hum";if="core.s",
</a/>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d/>;rt="simple.dev";if="core.ll",
</l/>;if="core.lb",
```

Figure 1: Binding Interface Example

4.1. Link List

The Link List interface is used to retrieve (GET) a list of resources on an origin server. The GET request SHOULD contain an Accept option with the application/link-format content format. However if the resource does not support any other form of content-format the Accept option MAY be elided.

Note: The use of an Accept option with application/link-format is recommended even though it is not strictly needed for the link list interface because this interface is extended by the batch and linked batch interfaces where different content-formats are possible.

The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on an origin server.

Link List is the base interface to provide gradual reveal of resources on a CoRE origin 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"
```

4.2. Batch

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

In addition, the Batch interface is an extension of the Link List interface and in consequence MUST support the same methods. For example, a GET with an Accept:application/link-format on a resource utilizing the batch interface will return the sub-resource links.

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" }],
}
```

4.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the origin server, a Linked Batch is dynamically controlled by a client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC8288] and the CoRE Link Format [RFC6690]. A request with a POST method and a content format of application/link-format simply appends new resource links to the collection. The links in the payload MUST reference a resource on the origin server with an absolute path. A DELETE request removes the entire collection. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /l/ and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.

```
Req: POST /l/ (Content-Format: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed

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

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

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

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

Req: DELETE /l/
Res: 2.02 Deleted
```

4.4. Sensor

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

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

```
Req: GET /s/humidity (Accept: text/plain)
Res: 2.05 Content (text/plain)
80

Req: GET /s/humidity (Accept: application/senml+json)
Res: 2.05 Content (application/senml+json)
{"e":[
    { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

4.5. Parameter

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

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

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

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

4.6. Read-only Parameter

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

The following example shows request for reading such a parameter.

```
Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
```

4.7. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or the actuator value can be updated (PUT). In addition, this interface allows the use of POST to change

the state of an actuator, for example to toggle between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to updated.

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

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

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

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

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

5. Security Considerations

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

6. IANA Considerations

This document registers the following CoRE Interface Description (if=) Link Target Attribute Values.

6.1. Link List

Attribute Value: core.ll

Description: The Link List interface is used to retrieve a list of resources on an origin server.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.2. Batch

Attribute Value: core.b

Description: The Batch interface is used to manipulate a collection of sub-resources at the same time.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.3. Linked Batch

Attribute Value: core.lb

Description: The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the origin server, a Linked Batch is dynamically controlled by a client.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.4. Sensor

Attribute Value: core.s

Description: The Sensor interface allows the value of a sensor resource to be read.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.5. Parameter

Attribute Value: core.p

Description: The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read or update.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.6. Read-only parameter

Attribute Value: core.rp

Description: The Read-only Parameter interface allows configuration parameters to be read but not updated.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.7. Actuator

Attribute Value: core.a

Description: The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read or the actuator value can be updated. In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle between its possible values.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

7. Acknowledgements

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

8. Changelog

Changes from -10 to 11:

- New Section 3.4 on Link Embedding
- Removed disused "Service discovery" terminology
- Removed wording referring to discontinued function set concept

Changes from -09 to -10:

- Section 1: Amendments to remove discussing properties.
- New author and editor added.

Changes from -08 to -09:

- Section 3.6: Modified to indicate that the entire collection resource is returned.
- General: Added editor's note with open issues.

Changes from -07 to -08:

- Section 3.3: Modified Accepts to Accept header option.
- Addressed the editor's note in Section 4.1 to clarify the use of the Accept option.

Changes from -06 to -07:

- Corrected Figure 1 sub-resource names e.g. tmp to temp and hum to humidity.
- Addressed the editor's note in Section 4.2.
- Removed section on function sets and profiles as agreed to at the IETF#97.

Changes from -05 to -06:

- Updated the abstract.
- Section 1: Updated introduction.
- Section 2: Alphabetised the order

- Section 2: Removed the collections definition in favour of the complete definition in the collections section.
- Removed section 3 on interfaces in favour of an updated definition in section 1.3.
- General: Changed interface type to interface description as that is the term defined in RFC6690.
- Removed section on future interfaces.
- Section 8: Updated IANA considerations.
- Added Appendix A to discuss current state of the art wrt to collections, function sets etc.

Changes from -04 to -05:

- Removed Link Bindings and Observe attributes. This functionality is now contained in I-D.ietf-core-dynlink.
- Hypermedia collections have been removed. This is covered in a new T2TRG draft.
- The WADL description has been removed.
- Fixed minor typos.
- Updated references.

Changes from -03 to -04:

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

Changes from -02 to -03:

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

Changes from -01 to -02:

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

Changes from -00 to WG Document -01

- Improvements to the Function Set section.

Changes from -05 to WG Document -00

- Updated the date and version.

Changes from -04 to -05

- Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.

Changes from -03 to -04

- Draft refresh

Changes from -02 to -03

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

Changes from -01 to -02

- Defined a Function Set and its guidelines.
- Added the Link List interface.
- Added the Linked Batch interface.
- Improved the WADL interface definition.
- Added a simple profile example.

9. References

9.1. Normative References

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

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

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

9.2. Informative References

[I-D.ietf-core-dynlink] Shelby, Z., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-04 (work in progress), September 2017.

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-13 (work in progress), March 2018.

[I-D.ietf-core-senml] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-13 (work in progress), March 2018.

[OIC-Core]
"OIC Resource Type Specification v1.1.0", 2016,
<https://openconnectivity.org/resources/specifications>.

[OIC-SmartHome]
"OIC Smart Home Device Specification v1.1.0", 2016,
<https://openconnectivity.org/resources/specifications>.

[OMA-TS-LWM2M]
"Lightweight Machine to Machine Technical Specification",
2016, <http://technical.openmobilealliance.org/Technical/
technical-information/release-program/current-releases/
oma-lightweightm2m-v1-0>.

[oneM2MTS0008]
"TS 0008 v1.3.2 CoAP Protocol Binding", 2016,
<http://www.onem2m.org/technical/published-documents>.

[oneM2MTS0023]
"TS 0023 v2.0.0 Home Appliances Information Model and
Mapping", 2016,
<http://www.onem2m.org/technical/published-documents>.

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

[RFC6573] Amundsen, M., "The Item and Collection Link Relations",
RFC 6573, DOI 10.17487/RFC6573, April 2012,
<https://www.rfc-editor.org/info/rfc6573>.

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

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

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

Appendix A. Current Usage of Interfaces and Function Sets

Editor's note: This appendix will be removed. It is only included for information.

This appendix analyses the current landscape with regards the definition and use of collections, interfaces and function sets/ profiles. This should be considered when considering the scope of this document.

In summary it can be seen that there is a lack of consistency of the definition and usage of interface description and function sets.

A.1. Constrained RESTful Environments (CoRE) Link Format (IETF)

[RFC6690] assumes that different deployments or application domains will define the appropriate REST Interface Descriptions along with Resource Types to make discovery meaningful. It highlights that collections are often used for these interfaces.

Whilst 3.2/[RFC6690] defines a new Interface Description 'if' attribute the procedures around it are about the naming of the interface not what information should be included in the documentation about the interface.

Function sets are not discussed.

A.2. CoRE Resource Directory (IETF)

[I-D.ietf-core-resource-directory] uses the concepts of collections, interfaces and function sets.

If defines a number of interfaces: discovery, registration, registration update, registration removal, read endpoint links, update endpoint links, registration request interface, removal request interface and lookup interface. However it does not assign an interface description identifier (if=) to these interfaces.

It does define a resource directory function set which specifies relevant content formats and interfaces to be used between a resource directory and endpoints. However it does not follow the format proposed by this document.

A.3. Open Connectivity Foundation (OCF)

The OIC Core Specification [OIC-Core] most closely aligns with the work in this specification. It makes use of interface descriptions as per [RFC6690] and has registered several interface identifiers

(https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#if-link-target-att-value). These interface descriptors are similar to those defined in this specification. From a high level perspective:

```
links list:   OCF (oic.if.ll) -> IETF (core.ll)
              Note: it's called "link list" in the IETF.
linked batch: OCF (oic.if.b) -> IETF (core.lb)
read-only:    OCF (oic.if.r) -> IETF (core.rp)
read-write:   OCF (oic.if.rw) -> IETF (core.p)
actuator:     OCF (oic.if.a) -> IETF (core.a)
sensor:       OCF (oic.if.s) -> IETF (core.s)
batch:        No OCF equivalent -> IETF (core.b)
```

Some of the OCF interfaces make use of collections.

The OIC Core specification does not use the concept of function sets. It does however discuss the concept of profiles. The OCF defines two sets of documents. The core specification documents such as [OIC-Core] and vertical profile specification documents which provide specific information for specific applications. The OIC Smart Home Device Specification [OIC-SmartHome] is one such specification. It provides information on the resource model, discovery and data types.

A.4. oneM2M

OneM2M describes a technology independent functional architecture [oneM2MTS0023]. In this archictecture the reference points between functional entities are called "interfaces". This usage does not match the [RFC6690] concept of interfaces. A more direct comparison is that of 10.2/[oneM2MTS0023] that defines basic procedures and resource type-specific procedures utilising REST type create, retrieve, update, delete, notify actions.

[oneM2MTS0023] does not refer to resource collections however does define "Group Management Procedures" in 10.2.7/[oneM2MTS0023]. It does allow bulk management of member resources.

[oneM2MTS0023] does not use the term "function set". [oneM2MTS0008] describes the binding with the CoAP protocol. In some respects this document provides a profile of the CoAP protocol in terms of the protocol elements that need to be supported. However it does not define any interface descriptions nor collections.

A.5. OMA LWM2M

[OMA-TS-LWM2M] utilises the concept of interfaces. It defines the following interfaces: Bootstrap, Client Registration, Device Management and Service Enablement and Information Reporting. It defines that these have a particular direction (Uplink/Downlink) and indicates the operations that may be applied to the interface (i.e. Request Bootstrap, Write, Delete, Register, Update, De-Register, Create, Read, Write, Delete, Execute, Write Attributes, Discover, Observe, Cancel Observation, Notify). It then further defines which objects may occur over the interface. In 6/[OMA-TS-LWM2M] resource model, identifier and data formats are described.

Whilst it does not formally describe the use of "collections" the use of a multiple resource TLV allows a hierarchy of resource/sub-resource.

It does not identify the interfaces through an Interface Description (if=) attribute.

It does not use the term function set. Informally the specification could be considered as a function set.

Note: It refers to draft-ietf-core-interfaces-00. It also makes use of the binding/observation attributes from draft-ietf-dynlink-00 but does not refer to that document.

Appendix B. Resource Profile example

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

Functions	Root Path	RT	IF
Device Description	/d	simple.dev	core.ll
Sensors	/s	simple.sen	core.b
Actuators	/a	simple.act	core.b

Table 3: Functional list of resources

Type	Path	RT	IF	Data Type
Name	/d/name	simple.dev.n	core.p	xsd:string
Model	/d/model	simple.dev.mdl	core.rp	xsd:string

Table 4: Device Description Resources

Type	Path	RT	IF	Data Type
Light	/s/light	simple.sen.lt	core.s	xsd:decimal (lux)
Humidity	/s/humidity	simple.sen.hum	core.s	xsd:decimal (%RH)
Temperature	/s/temp	simple.sen.tmp	core.s	xsd:decimal (degC)

Table 5: Sensor Resources

Type	Path	RT	IF	Data Type
LED	/a/{#}/led	simple.act.led	core.a	xsd:boolean

Table 6: Actuator Resources

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
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smartthings.com

Christian Groves
Australia

Email: cngroves.std@gmail.com

Julian Zhu
Huawei
No.127 Jinye Road, Huawei Base, High-Tech Development District
Xi'an, Shaanxi Province
China

Email: jintao.zhu@huawei.com

Bilhanan Silverajan (editor)
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

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

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

February 26, 2018

Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR
draft-ietf-core-links-json-10

Abstract

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

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

Table of Contents

1. Introduction 2
1.1. Objectives 3
1.2. Terminology 4
2. Web Links in JSON and CBOR 4
2.1. Background 4
2.2. Information Model 4
2.3. Additional Encoding Step for CBOR 6
2.4. Converting JSON or CBOR to Link-Format 8
2.5. Examples 9
2.5.1. Link Format to JSON Example 9
2.5.2. Link Format to CBOR Example 10
3. IANA Considerations 12
3.1. Media types 12
3.2. CoAP Content-Format Registration 13
4. Security Considerations 14
5. References 14
5.1. Normative References 14
5.2. Informative References 15
Appendix A. Reference implementation 16
Acknowledgements 19
Authors' Addresses 20

1. Introduction

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

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

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

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

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

1.1. Objectives

This specification has been designed based on the following objectives:

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

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

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

1.2. Terminology

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

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

CoAP: Constrained Application Protocol [RFC7252]

CBOR: Concise Binary Object Representation [RFC7049]

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

IoT: Internet of Things

JSON: JavaScript Object Notation [RFC8259]

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

2. Web Links in JSON and CBOR

2.1. Background

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

2.2. Information Model

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

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

We straightforwardly map:

- the collection of Web links to a JSON or CBOR 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 can be a string, a language-tagged string, a boolean, or an array of these, as described below.

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

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

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

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

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

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

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

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

Figure 1: CoRE Link Format Data Model (JSON)

2.3. Additional Encoding Step for CBOR

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

The substitution is defined in Table 1:

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

Table 1: Integer Encoding of common attribute names

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

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

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

Figure 2: CoRE Link Format Data Model (CBOR)

2.4. Converting JSON or CBOR to Link-Format

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

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

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

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

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

2.5. Examples

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

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

Figure 3: Example from page 15 of [RFC6690]

2.5.1. Link Format to JSON Example

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

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

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

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

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

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

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

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

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

2.5.2. Link Format to CBOR Example

This examples shows conversion from link format to CBOR format.

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

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

or, in hexadecimal (203 bytes):

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

```
      6d                          # text string(13 bytes)
         2f73656e736f72732f74
         656d70                   # "/sensors/temp"
      09                          # unsigned integer(value:9,"rt")
      6d                          # text string(13 bytes)
         74656d70657261747572
         652d63                   # "temperature-c"
      0a                          # unsigned integer(value:10,"if")
      66                          # text string(6 bytes)
         73656e736f72             # "sensor"
   a3                             # map(# data item pairs:3)
      01                          # unsigned integer(value:1,"href")
      6e                          # text string(14 bytes)
         2f73656e736f72732f6c
         69676874                 # "/sensors/light"
      09                          # unsigned integer(value:9,"rt")
      69                          # text string(9 bytes)
         6c696768742d6c7578       # "light-lux"
      0a                          # unsigned integer(value:10,"if")
      66                          # text string(6 bytes)
         73656e736f72             # "sensor"
   a3                             # map(# data item pairs:3)
      01                          # unsigned integer(value:1,"href")
      78 23                       # text string(35 bytes)
         687474703a2f2f777777
         2e6578616d706c652e63
         6f6d2f73656e736f7273
         2f74313233           # "http://www.example.com/sensors/t123"
      03                          # unsigned integer(value:3,"anchor")
      6d                          # text string(13 bytes)
         2f73656e736f72732f74
         656d70                   # "/sensors/temp"
      02                          # unsigned integer(value:2,"rel")
      6b                          # text string(11 bytes)
         646573637269626564279    # "describedby"
   a3                             # map(# data item pairs:3)
      01                          # unsigned integer(value:1,"href")
      62                          # text string(2 bytes)
         2f74                     # "/t"
      03                          # unsigned integer(value:3,"anchor")
      6d                          # text string(13 bytes)
         2f73656e736f72732f74
         656d70                   # "/sensors/temp"
      02                          # unsigned integer(value:2,"rel")
      69                          # text string(9 bytes)
         616c7465726e617465       # "alternate"
```

Figure 6: Web Links Encoded in CBOR

3. IANA Considerations

3.1. Media types

This specification registers the following additional Internet Media Types:

Type name: application

Subtype name: link-format+json

Required parameters: None

Optional parameters: None

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

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

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

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): TEXT

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

Intended usage: COMMON

Change controller: IESG

and

Type name: application

Subtype name: link-format+cbor

Required parameters: None

Optional parameters: None

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

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

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

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): CBOR

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

Intended usage: COMMON

Change controller: IESG

3.2. CoAP Content-Format Registration

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

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

Table 2: CoAP Content-Format IDs

4. Security Considerations

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

5. References

5.1. Normative References

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

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

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

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

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

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

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

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

5.2. Informative References

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

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

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

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

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

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

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

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

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

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

Appendix A. Reference implementation

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

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

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

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

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

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

```
     obs: 13,
       DATA

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

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

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

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

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

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

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

Acknowledgements

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

Hannes Tschofenig made many helpful suggestions for improving this document.

Authors' Addresses

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

Email: kepeng.lkp@alibaba-inc.com

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

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

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

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 20, 2018

G. Selander
J. Mattsson
F. Palombini
Ericsson AB
L. Seitz
RISE SICS

March 19, 2018

Object Security for Constrained RESTful Environments (OSCORE)
draft-ietf-core-object-security-11

Abstract

This document defines Object Security for Constrained RESTful Environments (OSCORE), a method for application-layer protection of the Constrained Application Protocol (CoAP), using CBOR Object Signing and Encryption (COSE). OSCORE provides end-to-end protection between endpoints communicating using CoAP or CoAP-mappable HTTP. OSCORE is designed for constrained nodes and networks supporting a range of proxy operations, including translation between different transport protocols.

Table of Contents

1. Introduction 3
1.1. Terminology 6
2. The CoAP Object-Security Option 6
3. The Security Context 7
3.1. Security Context Definition 7
3.2. Establishment of Security Context Parameters 9
3.3. Requirements on the Security Context Parameters 11
4. Protected Message Fields 12
4.1. CoAP Options 13
4.2. CoAP Header Fields and Payload 20
4.3. Signaling Messages 21
5. The COSE Object 22
5.1. Kid Context 23
5.2. Nonce 24
5.3. Plaintext 24
5.4. Additional Authenticated Data 25
6. OSCORE Header Compression 26
6.1. Encoding of the Object-Security Value 26
6.2. Encoding of the OSCORE Payload 28
6.3. Examples of Compressed COSE Objects 28
7. Sequence Numbers, Replay, Message Binding, and Freshness . . 30
7.1. Message Binding 30
7.2. AEAD Nonce Uniqueness 30
7.3. Freshness 31
7.4. Replay Protection 31
7.5. Losing Part of the Context State 32
8. Processing 33
8.1. Protecting the Request 33
8.2. Verifying the Request 34
8.3. Protecting the Response 35
8.4. Verifying the Response 36
9. Web Linking 37
10. Proxy and HTTP Operations 37
10.1. CoAP-to-CoAP Forwarding Proxy 38
10.2. HTTP Processing 38
10.3. HTTP-to-CoAP Translation Proxy 40
10.4. CoAP-to-HTTP Translation Proxy 41
11. Security Considerations 43
11.1. End-to-end protection 43

11.2. Security Context Establishment 44
11.3. Replay Protection 44
11.4. Cryptographic Considerations 44
11.5. Message Segmentation 45
11.6. Privacy Considerations 45
12. IANA Considerations 45
12.1. COSE Header Parameters Registry 46
12.2. CoAP Option Numbers Registry 46
12.3. CoAP Signaling Option Numbers Registry 46
12.4. Header Field Registrations 46
12.5. Media Type Registrations 47
12.6. CoAP Content-Formats Registry 49
13. References 49
13.1. Normative References 49
13.2. Informative References 51
Appendix A. Scenario Examples 52
A.1. Secure Access to Sensor 52
A.2. Secure Subscribe to Sensor 53
Appendix B. Deployment examples 55
B.1. Master Secret Used Once 55
B.2. Master Secret Used Multiple Times 55
B.3. Client Aliveness 56
Appendix C. Test Vectors 57
C.1. Test Vector 1: Key Derivation with Master Salt 57
C.2. Test Vector 2: Key Derivation without Master Salt 58
C.3. Test Vector 3: OSCORE Request, Client 59
C.4. Test Vector 4: OSCORE Request, Client 60
C.5. Test Vector 5: OSCORE Response, Server 61
C.6. Test Vector 6: OSCORE Response with Partial IV, Server . 62
Appendix D. Overview of Security Properties 64
D.1. Supporting Proxy Operations 64
D.2. Protected Message Fields 64
D.3. Uniqueness of (key, nonce) 65
D.4. Unprotected Message Fields 66
Appendix E. CDDL Summary 68
Acknowledgments 69
Authors' Addresses 69

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol, designed for constrained nodes and networks [RFC7228], and may be mapped from HTTP [RFC8075]. CoAP specifies the use of proxies for scalability and efficiency and references DTLS [RFC6347] for security. CoAP-to-CoAP, HTTP-to-CoAP, and CoAP-to-HTTP proxies require (D)TLS to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or

manipulate any part of, the message payload and metadata in transit between the endpoints. The proxy can also inject, delete, or reorder packets since they are no longer protected by (D)TLS.

This document defines the Object Security for Constrained RESTful Environments (OSCORE) security protocol, protecting CoAP and CoAP-mappable HTTP requests and responses end-to-end across intermediary nodes such as CoAP forward proxies and cross-protocol translators including HTTP-to-CoAP proxies [RFC8075]. In addition to the core CoAP features defined in [RFC7252], OSCORE supports Observe [RFC7641], Block-wise [RFC7959], No-Response [RFC7967], and PATCH and FETCH [RFC8132]. An analysis of end-to-end security for CoAP messages through some types of intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs]. OSCORE essentially protects the RESTful interactions; the request method, the requested resource, the message payload, etc. (see Section 4). OSCORE protects neither the CoAP Messaging Layer nor the CoAP Token which may change between the endpoints, and those are therefore processed as defined in [RFC7252]. Additionally, since the message formats for CoAP over unreliable transport [RFC7252] and for CoAP over reliable transport [RFC8323] differ only in terms of CoAP Messaging Layer, OSCORE can be applied to both unreliable and reliable transports (see Figure 1).

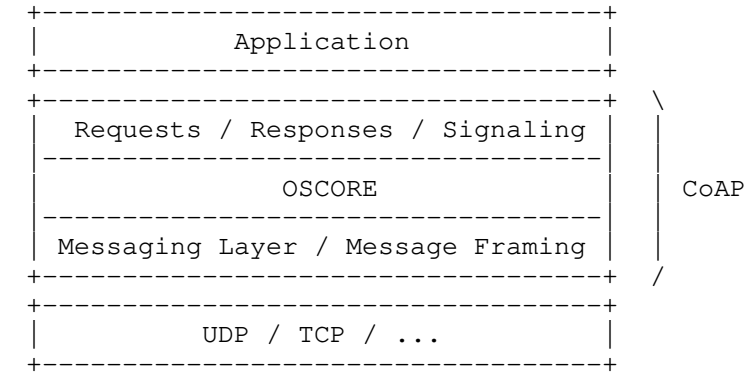


Figure 1: Abstract Layering of CoAP with OSCORE

OSCORE works in very constrained nodes and networks, thanks to its small message size and the restricted code and memory requirements in addition to what is required by CoAP. Examples of the use of OSCORE are given in Appendix A. OSCORE does not depend on underlying layers, and can be used anywhere where CoAP or HTTP can be used, including non-IP transports (e.g., [I-D.bormann-6lo-coap-802-15-ie]). OSCORE may be used together with (D)TLS over one or more hops in the end-to-end path, e.g. with HTTPS in one hop and with plain CoAP in another hop.

The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP or HTTP. The application decides the conditions for which OSCORE is required.

OSCORE uses pre-shared keys which may have been established out-of-band or with a key establishment protocol (see Section 3.2). The technical solution builds on CBOR Object Signing and Encryption (COSE) [RFC8152], providing end-to-end encryption, integrity, replay protection, and secure binding of response to request. A compressed version of COSE is used, as specified in Section 6. The use of OSCORE is signaled with the new Object-Security CoAP option or HTTP header field, defined in Section 2 and Section 10.3. The solution transforms a CoAP/HTTP message into an "OSCORE message" before sending, and vice versa after receiving. The OSCORE message is a CoAP/HTTP message related to the original message in the following way: the original CoAP/HTTP message is translated to CoAP (if not already in CoAP) and protected in a COSE object. The encrypted message fields of this COSE object are transported in the CoAP payload/HTTP body of the OSCORE message, and the Object-Security option/header field is included in the message. A sketch of an OSCORE message exchange in the case of the original message being CoAP is provided in Figure 2).

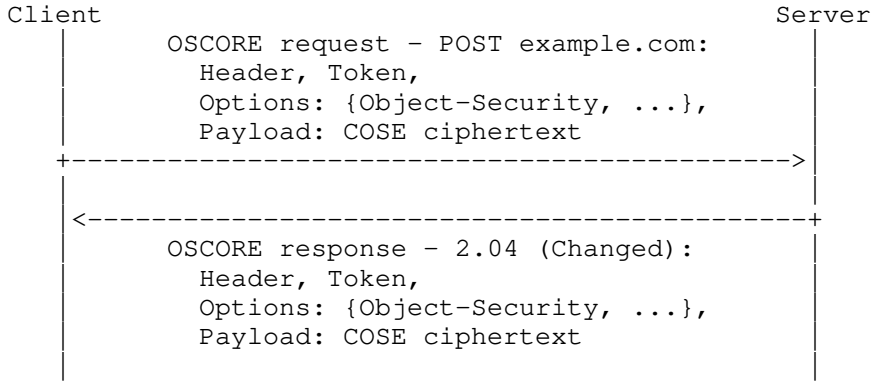


Figure 2: Sketch of CoAP with OSCORE

An implementation supporting this specification MAY implement only the client part, MAY implement only the server part, or MAY implement only one of the proxy parts. OSCORE is designed to protect as much information as possible while still allowing proxy operations (Section 10). It works with legacy CoAP-to-CoAP forward proxies [RFC7252], but an OSCORE-aware proxy will be more efficient. HTTP-to-CoAP proxies [RFC8075] and CoAP-to-HTTP proxies can also be used with OSCORE, as specified in Section 10.

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252], Observe [RFC7641], Block-wise [RFC7959], COSE [RFC8152], CBOR [RFC7049], CDDL [I-D.ietf-cbor-cddl] as summarized in Appendix E, and constrained environments [RFC7228].

The term "hop" is used to denote a particular leg in the end-to-end path. The concept "hop-by-hop" (as in "hop-by-hop encryption" or "hop-by-hop fragmentation") opposed to "end-to-end", is used in this document to indicate that the messages are processed accordingly in the intermediaries, rather than just forwarded to the next node.

The term "stop processing" is used throughout the document to denote that the message is not passed up to the CoAP Request/Response layer (see Figure 1).

The terms Common/Sender/Recipient Context, Master Secret/Salt, Sender ID/Key, Recipient ID/Key, and Common IV are defined in Section 3.1.

2. The CoAP Object-Security Option

The CoAP Object-Security option (see Figure 3, which extends Table 4 of [RFC7252]) indicates that the CoAP message is an OSCORE message and that it contains a compressed COSE object (see Section 5 and Section 6). The Object-Security option is critical, safe to forward, part of the cache key, and not repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD	x				Object-Security	(*)	0-255	(none)

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable
(*) See below.

Figure 3: The Object-Security Option

The Object-Security option includes the OSCORE flag bits (Section 6), the Sender Sequence Number and the Sender ID when present (Section 3). The detailed format and length is specified in Section 6. If the OSCORE flag bits is all zero (0x00) the Option

value SHALL be empty (Option Length = 0). An endpoint receiving a CoAP message without payload, that also contains an Object-Security option SHALL treat it as malformed and reject it.

A successful response to a request with the Object-Security option SHALL contain the Object-Security option. Whether error responses contain the Object-Security option depends on the error type (see Section 8).

A CoAP proxy SHOULD NOT cache a response to a request with an Object-Security option, since the response is only applicable to the original request (see Section 10.1). As the compressed COSE Object is included in the cache key, messages with the Object-Security option will never generate cache hits. For Max-Age processing, see Section 4.1.3.1.

3. The Security Context

OSCORE requires that client and server establish a shared security context used to process the COSE objects. OSCORE uses COSE with an Authenticated Encryption with Additional Data (AEAD, [RFC5116]) algorithm for protecting message data between a client and a server. In this section, we define the security context and how it is derived in client and server based on a shared secret and a key derivation function (KDF).

3.1. Security Context Definition

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCORE. For each endpoint, the security context is composed of a "Common Context", a "Sender Context", and a "Recipient Context".

The endpoints protect messages to send using the Sender Context and verify messages received using the Recipient Context, both contexts being derived from the Common Context and other data. Clients and servers need to be able to retrieve the correct security context to use.

An endpoint uses its Sender ID (SID) to derive its Sender Context, and the other endpoint uses the same ID, now called Recipient ID (RID), to derive its Recipient Context. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Thus, the two security contexts identified by the same IDs in the two endpoints are not the same, but they are partly mirrored. Retrieval and use of the security context are shown in Figure 4.

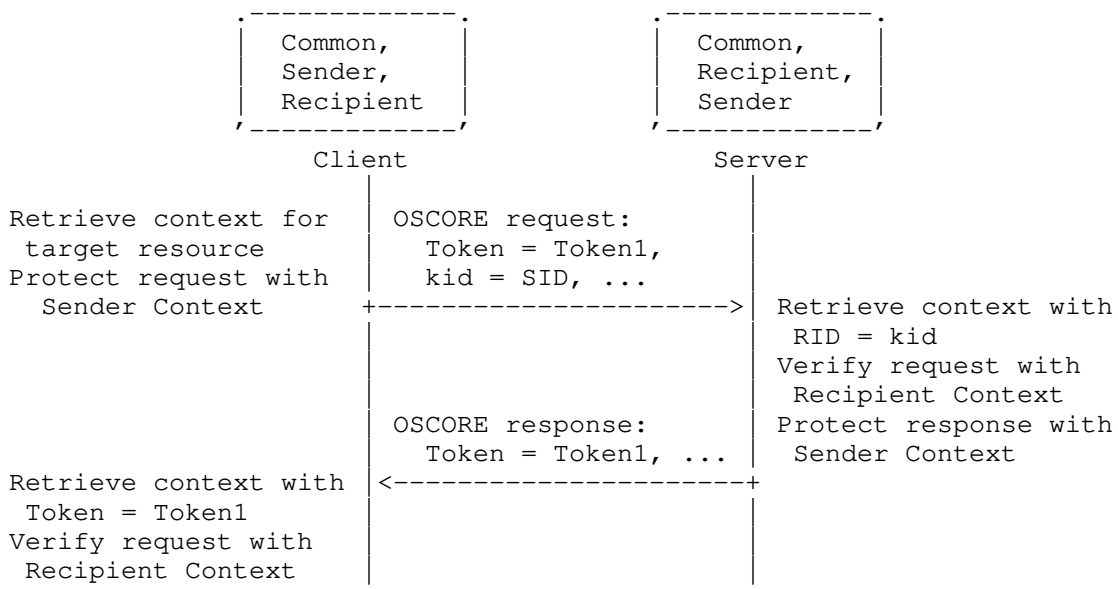


Figure 4: Retrieval and use of the Security Context

The Common Context contains the following parameters:

- AEAD Algorithm. The COSE AEAD algorithm to use for encryption.
- Key Derivation Function. The HMAC based HKDF [RFC5869] used to derive Sender Key, Recipient Key, and Common IV.
- Master Secret. Variable length, uniformly random byte string containing the key used to derive traffic keys and IVs.
- Master Salt. Variable length byte string containing the salt used to derive traffic keys and IVs.
- Common IV. Byte string derived from Master Secret and Master Salt. Length is determined by the AEAD Algorithm.

The Sender Context contains the following parameters:

- Sender ID. Byte string used to identify the Sender Context and to assure unique AEAD nonces. Maximum length is determined by the AEAD Algorithm.
- Sender Key. Byte string containing the symmetric key to protect messages to send. Derived from Common Context and Sender ID. Length is determined by the AEAD Algorithm.

- Sender Sequence Number. Non-negative integer used by the sender to protect requests and Observe notifications. Used as 'Partial IV' [RFC8152] to generate unique nonces for the AEAD. Maximum value is determined by the AEAD Algorithm.

The Recipient Context contains the following parameters:

- Recipient ID. Byte string used to identify the Recipient Context and to assure unique AEAD nonces. Maximum length is determined by the AEAD Algorithm.

- Recipient Key. Byte string containing the symmetric key to verify messages received. Derived from Common Context and Recipient ID. Length is determined by the AEAD Algorithm.

- Replay Window (Server only). The replay window to verify requests received.

All parameters except Sender Sequence Number and Replay Window are immutable once the security context is established. An endpoint may free up memory by not storing the Common IV, Sender Key, and Recipient Key, deriving them from the Master Key and Master Salt when needed. Alternatively, an endpoint may free up memory by not storing the Master Secret and Master Salt after the other parameters have been derived.

Endpoints MAY operate as both client and server and use the same security context for those roles. Independent of being client or server, the endpoint protects messages to send using its Sender Context, and verifies messages received using its Recipient Context. The endpoints MUST NOT change the Sender/Recipient ID when changing roles. In other words, changing the roles does not change the set of keys to be used.

3.2. Establishment of Security Context Parameters

The parameters in the security context are derived from a small set of input parameters. The following input parameters SHALL be pre-established:

- Master Secret

- Sender ID

- Recipient ID

The following input parameters MAY be pre-established. In case any of these parameters is not pre-established, the default value indicated below is used:

- AEAD Algorithm
  - Default is AES-CCM-16-64-128 (COSE algorithm encoding: 10)
- Master Salt
  - Default is the empty string
- Key Derivation Function (KDF)
  - Default is HKDF SHA-256
- Replay Window Type and Size
  - Default is DTLS-type replay protection with a window size of 32 [RFC6347]

All input parameters need to be known to and agreed on by both endpoints, but the replay window may be different in the two endpoints. The way the input parameters are pre-established, is application specific. The OSCORE profile of the ACE framework may be used to establish the necessary input parameters [I-D.ietf-ace-oscore-profile], or a key exchange protocol for providing forward secrecy. Other examples of deploying OSCORE are given in Appendix B.

3.2.1. Derivation of Sender Key, Recipient Key, and Common IV

The KDF MUST be one of the HMAC based HKDF [RFC5869] algorithms defined in COSE. HKDF SHA-256 is mandatory to implement. The security context parameters Sender Key, Recipient Key, and Common IV SHALL be derived from the input parameters using the HKDF, which consists of the composition of the HKDF-Extract and HKDF-Expand steps [RFC5869]:

```
output parameter = HKDF(salt, IKM, info, L)
```

where:

- salt is the Master Salt as defined above
- IKM is the Master Secret as defined above
- info is a CBOR array consisting of:

```
info = [
    id : bstr,
    alg_aead : int / tstr,
    type : tstr,
    L : uint
]
```

where:

- id is the Sender ID or Recipient ID when deriving keys and the empty string when deriving the Common IV. The encoding is described in Section 5.

- alg_aead is the AEAD Algorithm, encoded as defined in [RFC8152].

- type is "Key" or "IV". The label is an ASCII string, and does not include a trailing NUL byte.

- L is the size of the key/IV for the AEAD algorithm used, in bytes.

For example, if the algorithm AES-CCM-16-64-128 (see Section 10.2 in [RFC8152]) is used, the integer value for alg_aead is 10, the value for L is 16 for keys and 13 for the Common IV.

3.2.2. Initial Sequence Numbers and Replay Window

The Sender Sequence Number is initialized to 0. The supported types of replay protection and replay window length is application specific and depends on how OSCORE is transported, see Section 7.4. The default is DTLS-type replay protection with a window size of 32 initiated as described in Section 4.1.2.6 of [RFC6347].

3.3. Requirements on the Security Context Parameters

As collisions may lead to the loss of both confidentiality and integrity, Sender ID SHALL be unique in the set of all security contexts using the same Master Secret and Master Salt. When a trusted third party assigns identifiers (e.g., using [I-D.ietf-ace-oauth-authz]) or by using a protocol that allows the parties to negotiate locally unique identifiers in each endpoint, the Sender IDs can be very short. The maximum length of Sender ID in bytes equals the length of AEAD nonce minus 6. For AES-CCM-16-64-128 the maximum length of Sender ID is 7 bytes.

To simplify retrieval of the right Recipient Context, the Recipient ID SHOULD be unique in the sets of all Recipient Contexts used by an endpoint. If an endpoint has the same Recipient ID with different Recipient Contexts, i.e. the Recipient Contexts are derived from

different keying material, then the endpoint may need to try multiple times before finding the right security context associated to the Recipient ID. The Client MAY provide a 'kid context' parameter (Section 5.1) to help the Server find the right context.

While the triple (Master Secret, Master Salt, Sender ID) MUST be unique, the same Master Salt MAY be used with several Master Secrets and the same Master Secret MAY be used with several Master Salts.

4. Protected Message Fields

OSCORE transforms a CoAP message (which may have been generated from an HTTP message) into an OSCORE message, and vice versa. OSCORE protects as much of the original message as possible while still allowing certain proxy operations (see Section 10). This section defines how OSCORE protects the message fields and transfers them end-to-end between client and server (in any direction).

The remainder of this section and later sections discuss the behavior in terms of CoAP messages. If HTTP is used for a particular hop in the end-to-end path, then this section applies to the conceptual CoAP message that is mappable to/from the original HTTP message as discussed in Section 10. That is, an HTTP message is conceptually transformed to a CoAP message and then to an OSCORE message, and similarly in the reverse direction. An actual implementation might translate directly from HTTP to OSCORE without the intervening CoAP representation.

Protection of Signaling messages (Section 5 of [RFC8323]) is specified in Section 4.3. The other parts of this section target Request/Response messages.

Message fields of the CoAP message may be protected end-to-end between CoAP client and CoAP server in different ways:

- Class E: encrypted and integrity protected,
- Class I: integrity protected only, or
- Class U: unprotected.

The sending endpoint SHALL transfer Class E message fields in the ciphertext of the COSE object in the OSCORE message. The sending endpoint SHALL include Class I message fields in the Additional Authenticated Data (AAD) of the AEAD algorithm, allowing the receiving endpoint to detect if the value has changed in transfer. Class U message fields SHALL NOT be protected in transfer. Class I

and Class U message field values are transferred in the header or options part of the OSCORE message, which is visible to proxies.

Message fields not visible to proxies, i.e., transported in the ciphertext of the COSE object, are called "Inner" (Class E). Message fields transferred in the header or options part of the OSCORE message, which is visible to proxies, are called "Outer" (Class I or U). There are currently no Class I options defined.

An OSCORE message may contain both an Inner and an Outer instance of a certain CoAP message field. Inner message fields are intended for the receiving endpoint, whereas Outer message fields are used to enable proxy operations. Inner and Outer message fields are processed independently.

4.1. CoAP Options

A summary of how options are protected is shown in Figure 5. Note that some options may have both Inner and Outer message fields which are protected accordingly. The options which require special processing are labelled with asterisks.

No.	Name	E	U
1	If-Match	x	
3	Uri-Host		x
4	ETag	x	
5	If-None-Match	x	
6	Observe		*
7	Uri-Port		x
8	Location-Path	x	
TBD	Object-Security		*
11	Uri-Path	x	
12	Content-Format	x	
14	Max-Age	*	*
15	Uri-Query	x	
17	Accept	x	
20	Location-Query	x	
23	Block2	*	*
27	Block1	*	*
28	Size2	*	*
35	Proxy-Uri		*
39	Proxy-Scheme		x
60	Size1	*	*
258	No-Response	*	*

E = Encrypt and Integrity Protect (Inner)
U = Unprotected (Outer)
* = Special

Figure 5: Protection of CoAP Options

Options that are unknown or for which OSCORE processing is not defined SHALL be processed as class E (and no special processing). Specifications of new CoAP options SHOULD define how they are processed with OSCORE. A new COAP option SHOULD be of class E unless it requires proxy processing.

4.1.1. Inner Options

Inner option message fields (class E) are used to communicate directly with the other endpoint.

The sending endpoint SHALL write the Inner option message fields present in the original CoAP message into the plaintext of the COSE object (Section 5.3), and then remove the Inner option message fields from the OSCORE message.

The processing of Inner option message fields by the receiving endpoint is specified in Section 8.2 and Section 8.4.

4.1.2. Outer Options

Outer option message fields (Class U or I) are used to support proxy operations.

The sending endpoint SHALL include the Outer option message field present in the original message in the options part of the OSCORE message. All Outer option message fields, including Object-Security, SHALL be encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Outer option message field.

The processing of Outer options by the receiving endpoint is specified in Section 8.2 and Section 8.4.

A procedure for integrity-protection-only of Class I option message fields is specified in Section 5.4. Proxies MUST NOT change the order of option's occurrences, for options repeatable and of class I.

Note: There are currently no Class I option message fields defined.

4.1.3. Special Options

Some options require special processing, marked with an asterisk '*' in Figure 5; the processing is specified in this section.

4.1.3.1. Max-Age

An Inner Max-Age message field is used to indicate the maximum time a response may be cached by the client (as defined in [RFC7252]), end-to-end from the server to the client, taking into account that the option is not accessible to proxies. The Inner Max-Age SHALL be processed by OSCORE as specified in Section 4.1.1.

An Outer Max-Age message field is used to avoid unnecessary caching of OSCORE error responses at OSCORE unaware intermediary nodes. A server MAY set a Class U Max-Age message field with value zero to OSCORE error responses, which are described in Section 7.4, Section 8.2 and Section 8.4. Such message field is then processed according to Section 4.1.2.

Successful OSCORE responses do not need to include an Outer Max-Age option since the responses are non-cacheable by construction (see Section 4.2).

4.1.3.2. The Block Options

Block-wise [RFC7959] is an optional feature. An implementation MAY support [RFC7252] and the Object-Security option without supporting block-wise transfers. The Block options (Block1, Block2, Size1, Size2), when Inner message fields, provide secure message segmentation such that each segment can be verified. The Block options, when Outer message fields, enables hop-by-hop fragmentation of the OSCORE message. Inner and Outer block processing may have different performance properties depending on the underlying transport. The end-to-end integrity of the message can be verified both in case of Inner and Outer Block-wise transfers provided all blocks are received.

4.1.3.2.1. Inner Block Options

The sending CoAP endpoint MAY fragment a CoAP message as defined in [RFC7959] before the message is processed by OSCORE. In this case the Block options SHALL be processed by OSCORE as Inner options (Section 4.1.1). The receiving CoAP endpoint SHALL process the OSCORE message according to Section 4.1.1 before processing Block-wise as defined in [RFC7959].

4.1.3.2.2. Outer Block Options

Proxies MAY fragment an OSCORE message using [RFC7959], by introducing Block option message fields that are Outer (Section 4.1.2) and not generated by the sending endpoint. Note that the Outer Block options are neither encrypted nor integrity protected. As a consequence, a proxy can maliciously inject block fragments indefinitely, since the receiving endpoint needs to receive the last block (see [RFC7959]) to be able to compose the OSCORE message and verify its integrity. Therefore, applications supporting OSCORE and [RFC7959] MUST specify a security policy defining a maximum unfragmented message size (MAX_UNFRAGMENTED_SIZE) considering the maximum size of message which can be handled by the endpoints. Messages exceeding this size SHOULD be fragmented by the sending endpoint using Inner Block options (Section 4.1.3.2.1).

An endpoint receiving an OSCORE message with an Outer Block option SHALL first process this option according to [RFC7959], until all blocks of the OSCORE message have been received, or the cumulated message size of the blocks exceeds MAX_UNFRAGMENTED_SIZE. In the former case, the processing of the OSCORE message continues as defined in this document. In the latter case the message SHALL be discarded.

Because of encryption of Uri-Path and Uri-Query, messages to the same server may, from the point of view of a proxy, look like they also target the same resource. A proxy SHOULD mitigate a potential mix-up of blocks from concurrent requests to the same server, for example using the Request-Tag processing specified in Section 3.3.2 of [I-D.ietf-core-echo-request-tag].

4.1.3.3. Proxy-Uri

Proxy-Uri, when present, is split by OSCORE into class U options and class E options, which are processed accordingly. When Proxy-Uri is used in the original CoAP message, Uri-* are not present [RFC7252].

The sending endpoint SHALL first decompose the Proxy-Uri value of the original CoAP message into the Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path, and Uri-Query options (if present) according to Section 6.4 of [RFC7252].

Uri-Path and Uri-Query are class E options and SHALL be protected and processed as Inner options (Section 4.1.1).

The Proxy-Uri option of the OSCORE message SHALL be set to the composition of Proxy-Scheme, Uri-Host, and Uri-Port options (if present) as specified in Section 6.5 of [RFC7252], and processed as an Outer option of Class U (Section 4.1.2).

Note that replacing the Proxy-Uri value with the Proxy-Scheme and Uri-* options works by design for all CoAP URIs (see Section 6 of [RFC7252]). OSCORE-aware HTTP servers should not use the userinfo component of the HTTP URI (as defined in Section 3.2.1 of [RFC3986]), so that this type of replacement is possible in the presence of CoAP-to-HTTP proxies. In future documents specifying cross-protocol proxying behavior using different URI structures, it is expected that the authors will create Uri-* options that allow decomposing the Proxy-Uri, and specify in which OSCORE class they belong.

An example of how Proxy-Uri is processed is given here. Assume that the original CoAP message contains:

- Proxy-Uri = "coap://example.com/resource?q=1"

During OSCORE processing, Proxy-Uri is split into:

- Proxy-Scheme = "coap"
- Uri-Host = "example.com"
- Uri-Port = "5683"

- Uri-Path = "resource"

- Uri-Query = "q=1"

Uri-Path and Uri-Query follow the processing defined in Section 4.1.1, and are thus encrypted and transported in the COSE object. The remaining options are composed into the Proxy-Uri included in the options part of the OSCORE message, which has value:

- Proxy-Uri = "coap://example.com"

See Sections 6.1 and 12.6 of [RFC7252] for more information.

4.1.3.4. Observe

Observe [RFC7641] is an optional feature. An implementation MAY support [RFC7252] and the Object-Security option without supporting [RFC7641]. The Observe option as used here targets the requirements on forwarding of [I-D.hartke-core-e2e-security-reqs] (Section 2.2.1).

In order for an OSCORE-unaware proxy to support forwarding of Observe messages [RFC7641], there SHALL be an Outer Observe option, i.e., present in the options part of the OSCORE message. The processing of the CoAP Code for Observe messages is described in Section 4.2.

To secure the order of notifications, the client SHALL maintain a Notification Number for each Observation it registers. The Notification Number is a non-negative integer containing the largest Partial IV of the successfully received notifications for the associated Observe registration (see Section 7.4). The Notification Number is initialized to the Partial IV of the first successfully received notification response to the registration request. In contrast to [RFC7641], the received Partial IV MUST always be compared with the Notification Number, which thus MUST NOT be forgotten after 128 seconds. The client MAY ignore the Observe option value.

If the verification fails, the client SHALL stop processing the response.

The Observe option in the CoAP request may be legitimately removed by a proxy. If the Observe option is removed from a CoAP request by a proxy, then the server can still verify the request (as a non-Observe request), and produce a non-Observe response. If the OSCORE client receives a response to an Observe request without an Outer Observe value, then it MUST verify the response as a non-Observe response. If the OSCORE client receives a response to a non-Observe request

with an Outer Observe value, it stops processing the message, as specified in Section 8.4.

Clients can re-register observations to ensure that the observation is still active and establish freshness again ([RFC7641] Section 3.3.1). When an OSCORE observation is refreshed, not only the ETags, but also the partial IV (and thus the payload and Object-Security option) change. The server uses the new request's Partial IV as the 'request_piv' of new responses.

4.1.3.5. No-Response

No-Response is defined in [RFC7967]. Clients using No-Response MUST set both an Inner (Class E) and an Outer (Class U) No-Response option, with same value.

The Inner No-Response option is used to communicate to the server the client's disinterest in certain classes of responses to a particular request. The Inner No-Response SHALL be processed by OSCORE as specified in Section 4.1.1.

The Outer No-Response option is used to support proxy functionality, specifically to avoid error transmissions from proxies to clients, and to avoid bandwidth reduction to servers by proxies applying congestion control when not receiving responses. The Outer No-Response option is processed according to Section 4.1.2.

In particular, step 8 of Section 8.4 is applied to No-Response.

Applications should consider that a proxy may remove the Outer No-Response option from the request. Applications using No-Response can specify policies to deal with cases where servers receive an Inner No-Response option only, which may be the result of the request having traversed a No-Response unaware proxy, and update the processing in Section 8.4 accordingly. This avoids unnecessary error responses to clients and bandwidth reductions to servers, due to No-Response unaware proxies.

4.1.3.6. Object-Security

The Object-Security option is only defined to be present in OSCORE messages, as an indication that OSCORE processing have been performed. The content in the Object-Security option is neither encrypted nor integrity protected as a whole but some part of the content of this option is protected (see Section 5.4). "OSCORE within OSCORE" is not supported: If OSCORE processing detects an Object-Security option in the original CoAP message, then processing SHALL be stopped.

4.2. CoAP Header Fields and Payload

A summary of how the CoAP header fields and payload are protected is shown in Figure 6, including fields specific to CoAP over UDP and CoAP over TCP (marked accordingly in the table).

Field	E	U
Version (UDP)		x
Type (UDP)		x
Length (TCP)		x
Token Length		x
Code	x	
Message ID (UDP)		x
Token		x
Payload	x	

E = Encrypt and Integrity Protect (Inner)
U = Unprotected (Outer)

Figure 6: Protection of CoAP Header Fields and Payload

Most CoAP Header fields (i.e. the message fields in the fixed 4-byte header) are required to be read and/or changed by CoAP proxies and thus cannot in general be protected end-to-end between the endpoints. As mentioned in Section 1, OSCORE protects the CoAP Request/Response layer only, and not the Messaging Layer (Section 2 of [RFC7252]), so fields such as Type and Message ID are not protected with OSCORE.

The CoAP Header field Code is protected by OSCORE. Code SHALL be encrypted and integrity protected (Class E) to prevent an intermediary from eavesdropping on or manipulating the Code (e.g., changing from GET to DELETE).

The sending endpoint SHALL write the Code of the original CoAP message into the plaintext of the COSE object (see Section 5.3). After that, the Outer Code of the OSCORE message SHALL be set to 0.02 (POST) for requests without Observe option, to 0.05 (FETCH) for requests with Observe option, and to 2.04 (Changed) for responses. Using FETCH with Observe allows OSCORE to be compliant with the Observe processing in OSCORE-unaware proxies. The choice of POST and FETCH [RFC8132] allows all OSCORE messages to have payload.

The receiving endpoint SHALL discard the Code in the OSCORE message and write the Code of the plaintext in the COSE object (Section 5.3) into the decrypted CoAP message.

The other currently defined CoAP Header fields are Unprotected (Class U). The sending endpoint SHALL write all other header fields of the original message into the header of the OSCORE message. The receiving endpoint SHALL write the header fields from the received OSCORE message into the header of the decrypted CoAP message.

The CoAP Payload, if present in the original CoAP message, SHALL be encrypted and integrity protected and is thus an Inner message field. The sending endpoint writes the payload of the original CoAP message into the plaintext (Section 5.3) input to the COSE object. The receiving endpoint verifies and decrypts the COSE object, and recreates the payload of the original CoAP message.

4.3. Signaling Messages

Signaling messages (CoAP Code 7.00-7.31) were introduced to exchange information related to an underlying transport connection in the specific case of CoAP over reliable transports [RFC8323].

OSCORE MAY be used to protect Signaling if the endpoints for OSCORE coincide with the endpoints for the signaling message. If OSCORE is used to protect Signaling then:

- To comply with [RFC8323], an initial empty CSM message SHALL be sent. The subsequent signaling message SHALL be protected.

- Signaling messages SHALL be protected as CoAP Request messages, except in the case the Signaling message is a response to a previous Signaling message, in which case it SHALL be protected as a CoAP Response message. For example, 7.02 (Ping) is protected as a CoAP Request and 7.03 (Pong) as a CoAP response.

- The Outer Code for Signaling messages SHALL be set to 0.02 (POST), unless it is a response to a previous Signaling message, in which case it SHALL be set to 2.04 (Changed).

- All Signaling options, except the Object-Security option, SHALL be Inner (Class E).

NOTE: Option numbers for Signaling messages are specific to the CoAP Code (see Section 5.2 of [RFC8323]).

If OSCORE is not used to protect Signaling, Signaling messages SHALL be unaltered by OSCORE.

5. The COSE Object

This section defines how to use COSE [RFC8152] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Additional Data (AEAD) algorithm. The key lengths, IV length, nonce length, and maximum Sender Sequence Number are algorithm dependent.

The AEAD algorithm AES-CCM-16-64-128 defined in Section 10.2 of [RFC8152] is mandatory to implement. For AES-CCM-16-64-128 the length of Sender Key and Recipient Key is 128 bits, the length of nonce and Common IV is 13 bytes. The maximum Sender Sequence Number is specified in Section 11.

As specified in [RFC5116], plaintext denotes the data that is to be encrypted and integrity protected, and Additional Authenticated Data (AAD) denotes the data that is to be integrity protected only.

The COSE Object SHALL be a COSE_Encrypt0 object with fields defined as follows

- The 'protected' field is empty.
- The 'unprotected' field includes:
  - The 'Partial IV' parameter. The value is set to the Sender Sequence Number. All leading zeroes SHALL be removed when encoding the Partial IV, except in the case of value 0 which is encoded to the byte string 0x00. This parameter SHALL be present in requests. In case of Observe (Section 4.1.3.4) the Partial IV SHALL be present in responses, and otherwise the Partial IV will not typically be present in responses. (A non-Observe example where the Partial IV is included in a response is provided in Section 7.5.2.)
  - The 'kid' parameter. The value is set to the Sender ID. This parameter SHALL be present in requests and will not typically be present in responses. An example where the Sender ID is included in a response is the extension of OSCORE to group communication [I-D.ietf-core-oscore-groupcomm].
  - Optionally, a 'kid context' parameter as defined in Section 5.1. This parameter MAY be present in requests and SHALL NOT be present in responses.
- The 'ciphertext' field is computed from the secret key (Sender Key or Recipient Key), AEAD nonce (see Section 5.2), plaintext (see

Section 5.3), and the Additional Authenticated Data (AAD) (see Section 5.4) following Section 5.2 of [RFC8152].

The encryption process is described in Section 5.3 of [RFC8152].

5.1. Kid Context

For certain use cases, e.g. deployments where the same kid is used with multiple contexts, it is necessary or favorable for the sender to provide an additional identifier of the security material to use, in order for the receiver to retrieve or establish the correct key. The kid context parameter is used to provide such additional input. The kid context and kid are used to determine the security context, or to establish the necessary input parameters to derive the security context (see Section 3.2). The application defines how this is done.

The kid context is implicitly integrity protected, as manipulation that leads to the wrong key (or no key) being retrieved which results in an error, as described in Section 8.2.

A summary of the COSE header parameter kid context defined above can be found in Figure 7.

Some examples of relevant uses of kid context are the following:

- If the client has an identifier in some other namespace which can be used by the server to retrieve or establish the security context, then that identifier can be used as kid context. The kid context may be used as Master Salt (Section 3.1) for additional entropy of the security contexts (see for example Appendix B.2 or [I-D.ietf-6tisch-minimal-security]).

- In case of a group communication scenario [I-D.ietf-core-oscore-groupcomm], if the server belongs to multiple groups, then a group identifier can be used as kid context to enable the server to find the right security context.

name	label	value type	value registry	description
kid context	kidctx	bstr		Identifies the kid context

Figure 7: Additional common header parameter for the COSE object

5.2. Nonce

The AEAD nonce is constructed in the following way (see Figure 8):

1. left-padding the Partial IV (in network byte order) with zeroes to exactly 5 bytes,

2. left-padding the (Sender) ID of the endpoint that generated the Partial IV (in network byte order) with zeroes to exactly nonce length - 6 bytes,

3. concatenating the size of the ID (S) with the padded ID and the padded Partial IV,

4. and then XORing with the Common IV.

Note that in this specification only algorithms that use nonces equal or greater than 7 bytes are supported. The nonce construction with S, ID of PIV generator, and Partial IV together with endpoint unique IDs and encryption keys make it easy to verify that the nonces used with a specific key will be unique.

When Observe is not used, the request and the response may use the same nonce. In this way, the Partial IV does not have to be sent in responses, which reduces the size. For processing instructions see Section 8.

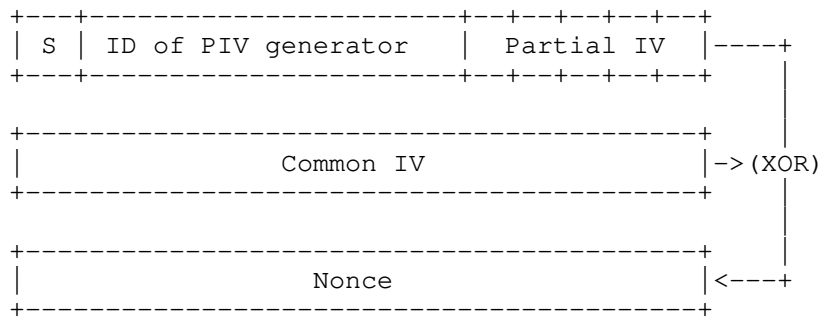


Figure 8: AEAD Nonce Formation

5.3. Plaintext

The plaintext is formatted as a CoAP message without Header (see Figure 9) consisting of:

- the Code of the original CoAP message as defined in Section 3 of [RFC7252]; and

- all Inner option message fields (see Section 4.1.1) present in the original CoAP message (see Section 4.1). The options are encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Class E option; and

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

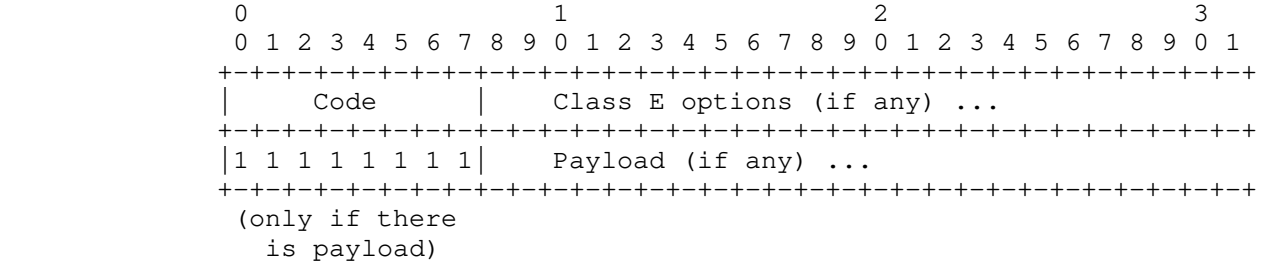


Figure 9: Plaintext

NOTE: The plaintext contains all CoAP data that needs to be encrypted end-to-end between the endpoints.

5.4. Additional Authenticated Data

The external_aad SHALL be a CBOR array as defined below:

```
external_aad = [
   oscore_version : uint,
   algorithms : [ alg_aead : int / tstr ],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr
]
```

where:

- oscore_version: contains the OSCORE version number. Implementations of this specification MUST set this field to 1. Other values are reserved for future versions.

- algorithms: contains (for extensibility) an array of algorithms, according to this specification only containing alg_aead.

- alg_aead: contains the AEAD Algorithm from the security context used for the exchange (see Section 3.1).

- request_kid: contains the value of the 'kid' in the COSE object of the request (see Section 5).

- request_piv: contains the value of the 'Partial IV' in the COSE object of the request (see Section 5).

- options: contains the Class I options (see Section 4.1.2) present in the original CoAP message encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of class I option.

NOTE: The format of the external_aad is for simplicity the same for requests and responses, although some parameters, e.g. request_kid need not be integrity protected in the requests.

6. OSCORE Header Compression

The Concise Binary Object Representation (CBOR) [RFC7049] combines very small message sizes with extensibility. The CBOR Object Signing and Encryption (COSE) [RFC8152] uses CBOR to create compact encoding of signed and encrypted data. COSE is however constructed to support a large number of different stateless use cases, and is not fully optimized for use as a stateful security protocol, leading to a larger than necessary message expansion. In this section, we define a stateless header compression mechanism, simply removing redundant information from the COSE objects, which significantly reduces the per-packet overhead. The result of applying this mechanism to a COSE object is called the "compressed COSE object".

The COSE_Encrypt0 object used in OSCORE is transported in the Object-Security option and in the Payload. The Payload contains the Ciphertext and the headers of the COSE object are compactly encoded as described in the next section.

6.1. Encoding of the Object-Security Value

The value of the Object-Security option SHALL contain the OSCORE flag bits, the Partial IV parameter, the kid context parameter (length and value), and the kid parameter as follows:

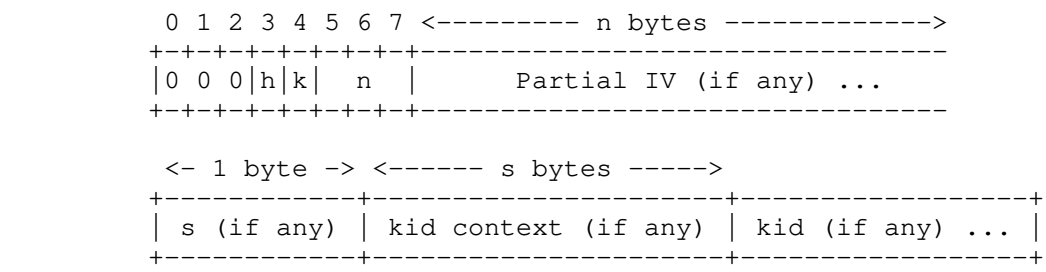


Figure 10: Object-Security Value

- The first byte of flag bits encodes the following set of flags and the length of the Partial IV parameter:
  - The three least significant bits encode the Partial IV length n. If n = 0 then the Partial IV is not present in the compressed COSE object. The values n = 6 and n = 7 are reserved.
  - The fourth least significant bit is the kid flag, k: it is set to 1 if the kid is present in the compressed COSE object.
  - The fifth least significant bit is the kid context flag, h: it is set to 1 if the compressed COSE object contains a kid context (see Section 5.1).
  - The sixth to eighth least significant bits are reserved for future use. These bits SHALL be set to zero when not in use. According to this specification, if any of these bits are set to 1 the message is considered to be malformed and decompression fails as specified in item 3 of Section 8.2.
- The following n bytes encode the value of the Partial IV, if the Partial IV is present (n > 0).
- The following 1 byte encode the length of the kid context (Section 5.1) s, if the kid context flag is set (h = 1).
- The following s bytes encode the kid context, if the kid context flag is set (h = 1).
- The remaining bytes encode the value of the kid, if the kid is present (k = 1).

Note that the kid MUST be the last field of the object-security value, even in case reserved bits are used and additional fields are added to it.

The length of the Object-Security option thus depends on the presence and length of Partial IV, kid context, kid, as specified in this section, and on the presence and length of the other parameters, as defined in the separate documents.

6.2. Encoding of the OSCORE Payload

The payload of the OSCORE message SHALL encode the ciphertext of the COSE object.

6.3. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples for requests and responses. The examples assume the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the full CoAP unprotected message, as well as the full security context, is not reported in the examples, but only the input necessary to the compression mechanism, i.e. the COSE_Encrypt0 object. The output is the compressed COSE object as defined in Section 6, divided into two parts, since the object is transported in two CoAP fields: Object-Security option value and CoAP payload.

6.3.1. Examples: Requests

1. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25 and Partial IV = 0x05

Before compression (24 bytes):

```
[
h'',
{ 4:h'25', 6:h'05' },
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (17 bytes):

Flag byte: 0b00001001 = 0x09

Option Value: 09 05 25 (3 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

2. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string and Partial IV = 0x00

Before compression (23 bytes):

```
[
h'',
{ 4:h'', 6:h'00' },
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (16 bytes):

Flag byte: 0b00001001 = 0x09

Option Value: 09 00 (2 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

3. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string, Partial IV = 0x05, and kid context = 0x44616c656b

Before compression (30 bytes):

```
[
h'',
{ 4:h'', 6:h'05', 8:h'44616c656b' },
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (22 bytes):

Flag byte: 0b00011001 = 0x19

Option Value: 19 05 05 44 61 6c 65 6b (8 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

6.3.2. Example: Response (without Observe)

1. Response not including an Observe option, with ciphertext = 0xaea0155667924dff8a24e4cb35b9

Before compression (18 bytes):

```
[
h'',
{},
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (14 bytes):

Flag byte: 0b00000000 = 0x00

Option Value: (0 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

6.3.3. Example: Response (with Observe)

1. Response including an Observe option, with ciphertext = 0xaea0155667924dff8a24e4cb35b9 and Partial IV = 0x07

Before compression (21 bytes):

```
[
h'',
{ 6:h'07' },
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (16 bytes):

Flag byte: 0b00000001 = 0x01

Option Value: 01 07 (2 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

7. Sequence Numbers, Replay, Message Binding, and Freshness

7.1. Message Binding

In order to prevent response delay and mismatch attacks [I-D.mattsson-core-coap-actuators] from on-path attackers and compromised proxies, OSCORE binds responses to the requests by including the kid and Partial IV of the request in the AAD of the response. The server therefore needs to store the kid and Partial IV of the request until all responses have been sent.

7.2. AEAD Nonce Uniqueness

An AEAD nonce MUST NOT be used more than once per AEAD key. In order to assure unique nonces, each Sender Context contains a Sender Sequence Number used to protect requests, and - in case of Observe - responses. If messages are processed concurrently, the operation of reading and increasing the Sender Sequence Number MUST be atomic.

The maximum Sender Sequence Number is algorithm dependent (see Section 11), and SHALL be less than 2^40. If the Sender Sequence

Number exceeds the maximum, the endpoint MUST NOT process any more messages with the given Sender Context. The endpoint SHOULD acquire a new security context (and consequently inform the other endpoint) before this happens. The latter is out of scope of this document.

7.3. Freshness

For requests, OSCORE provides only the guarantee that the request is not older than the security context. For applications having stronger demands on request freshness (e.g., control of actuators), OSCORE needs to be augmented with mechanisms providing freshness, for example as specified in [I-D.ietf-core-echo-request-tag].

For responses, the message binding guarantees that a response is not older than its request. For responses without Observe, this gives strong absolute freshness. For responses with Observe, the absolute freshness gets weaker with time, and it is RECOMMENDED that the client regularly re-register the observation.

For requests, and responses with Observe, OSCORE also provides relative freshness in the sense that the received Partial IV allows a recipient to determine the relative order of responses.

7.4. Replay Protection

In order to protect from replay of requests, the server's Recipient Context includes a Replay Window. A server SHALL verify that a Partial IV received in the COSE object has not been received before. If this verification fails the server SHALL stop processing the message, and MAY optionally respond with a 4.01 Unauthorized error message. Also, the server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the "Replay detected" string. The size and type of the Replay Window depends on the use case and the protocol with which the OSCORE message is transported. In case of reliable and ordered transport from endpoint to endpoint, e.g. TCP, the server MAY just store the last received Partial IV and require that newly received Partial IVs equals the last received Partial IV + 1. However, in case of mixed reliable and unreliable transports and where messages may be lost, such a replay mechanism may be too restrictive and the default replay window be more suitable (see Section 3.2.2).

Responses to non-Observe requests are protected against replay as they are cryptographically bound to the request.

In the case of Observe, a client receiving a notification SHALL verify that the Partial IV of a received notification is greater than the Notification Number bound to that Observe registration. If the

verification fails, the client SHALL stop processing the response. If the verification succeeds, the client SHALL overwrite the corresponding Notification Number with the received Partial IV.

If messages are processed concurrently, the Partial IV needs to be validated a second time after decryption and before updating the replay protection data. The operation of validating the Partial IV and updating the replay protection data MUST be atomic.

7.5. Losing Part of the Context State

To prevent reuse of the AEAD nonce with the same key, or from accepting replayed messages, an endpoint needs to handle the situation of losing rapidly changing parts of the context, such as the request Token, Sender Sequence Number, Replay Window, and Notification Numbers. These are typically stored in RAM and therefore lost in the case of an unplanned reboot.

After boot, an endpoint MAY reject to use pre-existing security contexts, and MAY establish a new security context with each endpoint it communicates with. However, establishing a fresh security context may have a non-negligible cost in terms of, e.g., power consumption.

After boot, an endpoint MAY use a partly persistently stored security context, but then the endpoint MUST NOT reuse a previous Sender Sequence Number and MUST NOT accept previously accepted messages. Some ways to achieve this are described in the following sections.

7.5.1. Sequence Number

To prevent reuse of Sender Sequence Numbers, an endpoint MAY perform the following procedure during normal operations:

- Each time the Sender Sequence Number is evenly divisible by K, where K is a positive integer, store the Sender Sequence Number in persistent memory. After boot, the endpoint initiates the Sender Sequence Number to the value stored in persistent memory + K - 1. Storing to persistent memory can be costly. The value K gives a trade-off between the number of storage operations and efficient use of Sender Sequence Numbers.

7.5.2. Replay Window

To prevent accepting replay of previously received requests, the server MAY perform the following procedure after boot:

- For each stored security context, the first time after boot the server receives an OSCORE request, the server responds with the

Echo option [I-D.ietf-core-echo-request-tag] to get a request with verifiable freshness. The server MUST use its Partial IV when generating the AEAD nonce and MUST include the Partial IV in the response.

If the server using the Echo option can verify a second request as fresh, then the Partial IV of the second request is set as the lower limit of the replay window.

7.5.3. Replay Protection of Observe Notifications

To prevent accepting replay of previously received notification responses, the client MAY perform the following procedure after boot:

- The client rejects notifications bound to the earlier registration, removes all Notification Numbers and re-registers using Observe.

8. Processing

This section describes the OSCORE message processing.

8.1. Protecting the Request

Given a CoAP request, the client SHALL perform the following steps to create an OSCORE request:

1. Retrieve the Sender Context associated with the target resource.

2. Compose the Additional Authenticated Data and the plaintext, as described in Section 5.4 and Section 5.3.

3. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV (Sender Sequence Number in network byte order) as described in Section 5.2 and (in one atomic operation, see Section 7.2) increment the Sender Sequence Number by one.

4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 6.

5. Format the OSCORE message according to Section 4. The Object-Security option is added (see Section 4.1.2).

6. Store the association Token - Security Context, in order to be able to find the Recipient Context from the Token in the response.

8.2. Verifying the Request

A server receiving a request containing the Object-Security option SHALL perform the following steps:

1. Process Outer Block options according to [RFC7959], until all blocks of the request have been received (see Section 4.1.3.2).

2. Discard the message Code and all non-special Inner option message fields (marked with 'x' in column E of Figure 5) present in the received message. For example, an If-Match Outer option is discarded, but an Uri-Host Outer option is not discarded.

3. Decompress the COSE Object (Section 6) and retrieve the Recipient Context associated with the Recipient ID in the 'kid' parameter. If either the decompression or the COSE message fails to decode, or the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, then the server SHALL stop processing the request. If:

   * either the decompression or the COSE message fails to decode, the server MAY respond with a 4.02 Bad Option error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload SHOULD contain the string "Failed to decode COSE".

   * the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, the server MAY respond with a 4.01 Unauthorized error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload SHOULD contain the string "Security context not found".

4. Verify the 'Partial IV' parameter using the Replay Window, as described in Section 7.4.

5. Compose the Additional Authenticated Data, as described in Section 5.4.

6. Compute the AEAD nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.

7. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.)

   * If decryption fails, the server MUST stop processing the request and MAY respond with a 4.00 Bad Request error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the "Decryption failed" string.

   * If decryption succeeds, update the Replay Window, as described in Section 7.

8. For each decrypted option, check if the option is also present as an Outer option: if it is, discard the Outer. For example: the message contains a Max-Age Inner and a Max-Age Outer option. The Outer Max-Age is discarded.

9. Add decrypted code, options and payload to the decrypted request. The Object-Security option is removed.

10. The decrypted CoAP request is processed according to [RFC7252].

8.3. Protecting the Response

If a CoAP response is generated in response to an OSCORE request, the server SHALL perform the following steps to create an OSCORE response. Note that CoAP error responses derived from CoAP processing (point 10. in Section 8.2) are protected, as well as successful CoAP responses, while the OSCORE errors (point 3, 4, and 7 in Section 8.2) do not follow the processing below, but are sent as simple CoAP responses, without OSCORE processing.

1. Retrieve the Sender Context in the Security Context used to verify the request.

2. Compose the Additional Authenticated Data and the plaintext, as described in Section 5.4 and Section 5.3.

3. Compute the AEAD nonce

   * If Observe is used, compute the nonce from the Sender ID, Common IV, and Partial IV (Sender Sequence Number in network byte order). Then (in one atomic operation, see Section 7.2) increment the Sender Sequence Number by one.

   * If Observe is not used, either the nonce from the request is used or a new Partial IV is used (see bullet on 'Partial IV' in Section 5).

4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 6. If the AEAD nonce was

constructed from a new Partial IV, this Partial IV MUST be included in the message. If the AEAD nonce from the request was used, the Partial IV MUST NOT be included in the message.

5. Format the OSCORE message according to Section 4. The Object-Security option is added (see Section 4.1.2).

8.4. Verifying the Response

A client receiving a response containing the Object-Security option SHALL perform the following steps:

1. Process Outer Block options according to [RFC7959], until all blocks of the OSCORE message have been received (see Section 4.1.3.2).

2. Discard the message Code and all non-special Class E options from the message. For example, ETag Outer option is discarded, Max-Age Outer option is not discarded.

3. Retrieve the Recipient Context associated with the Token. Decompress the COSE Object (Section 6). If either the decompression or the COSE message fails to decode, then go to 11.

4. For Observe notifications, verify the received 'Partial IV' parameter against the corresponding Notification Number as described in Section 7.4. If the client receives a notification for which no Observe request was sent, then go to 11.

5. Compose the Additional Authenticated Data, as described in Section 5.4.

6. Compute the AEAD nonce

    1. If the Observe option and the Partial IV are not present in the response, the nonce from the request is used.

    2. If the Observe option is present in the response, and the Partial IV is not present in the response, then go to 11.

    3. If the Partial IV is present in the response, compute the nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.

7. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.)

   * If decryption fails, then go to 11.

   * If decryption succeeds and Observe is used, update the corresponding Notification Number, as described in Section 7.

8. For each decrypted option, check if the option is also present as an Outer option: if it is, discard the Outer. For example: the message contains a Max-Age Inner and a Max-Age Outer option. The Outer Max-Age is discarded.

9. Add decrypted code, options and payload to the decrypted request. The Object-Security option is removed.

10. The decrypted CoAP response is processed according to [RFC7252].

11. In case any of the previous erroneous conditions apply: the client SHALL stop processing the response.

An error condition occurring while processing a response in an observation does not cancel the observation. A client MUST NOT react to failure in step 7 by re-registering the observation immediately.

9. Web Linking

The use of OSCORE MAY be indicated by a target attribute "osc" in a web link [RFC8288] to a resource. This attribute is a hint indicating that the destination of that link is to be accessed using OSCORE. Note that this is simply a hint, it does not include any security context material or any other information required to run OSCORE.

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

10. Proxy and HTTP Operations

RFC 7252 defines operations for a CoAP-to-CoAP proxy (see Section 5.7 of [RFC7252]) and for proxying between CoAP and HTTP (Section 10 of [RFC7252]). A more detailed description of the HTTP-to-CoAP mapping is provided by [RFC8075]. This section describes the operations of OSCORE-aware proxies.

10.1. CoAP-to-CoAP Forwarding Proxy

OSCORE is designed to work with legacy CoAP-to-CoAP forward proxies [RFC7252], but OSCORE-aware proxies MAY provide certain simplifications as specified in this section.

Security requirements for forwarding are presented in Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs]. OSCORE complies with the extended security requirements also addressing Block-wise [RFC7959] and CoAP-mappable HTTP. In particular caching is disabled since the CoAP response is only applicable to the original CoAP request. An OSCORE-aware proxy SHALL NOT cache a response to a request with an Object-Security option. As a consequence, the search for cache hits and CoAP freshness/Max-Age processing can be omitted.

Proxy processing of the (Outer) Proxy-Uri option is as defined in [RFC7252].

Proxy processing of the (Outer) Block options is as defined in [RFC7959].

Proxy processing of the (Outer) Observe option is as defined in [RFC7641]. OSCORE-aware proxies MAY look at the Partial IV value instead of the Outer Observe option.

10.2. HTTP Processing

In order to use OSCORE over HTTP hops, a node needs to be able to map HTTP messages to CoAP messages (see [RFC8075]), and to apply OSCORE to CoAP messages (as defined in this document).

For this purpose, this specification defines a new HTTP header field named CoAP-Object-Security, see Section 12.4. The CoAP-Object-Security header field is only used in POST requests and 200 (OK) responses, i.e. essentially using HTTP as a transport of an encrypted CoAP mappable message contained in the payload.

The header field is neither appropriate to list in the Connection header field (see Section 6.1 of [RFC7230]), nor in a Vary response header field (see Section 7.1.4 of [RFC7231]), nor allowed in trailers (see Section 4.1 of [RFC7230]).

[Ed. Note: Reconsider use of Vary]

Intermediaries cannot insert, delete, or modify the field's value without being detected. The header field is not preserved across redirects.

[Ed. Note: Reconsider support for redirects]

Using the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], including the following core ABNF syntax rules defined by that specification: ALPHA (letters) and DIGIT (decimal digits), the CoAP-Object-Security header field is as follows.

```
base64-char = ALPHA / DIGIT / "_" / "-"

CoAP-Object-Security = 2*base64-char
```

A sending endpoint uses [RFC8075] to translate an HTTP message into a CoAP message. It then protects the message with OSCORE processing, and adds the Object-Security option (as defined in this document). Then, the endpoint maps the resulting CoAP message to an HTTP message that includes the HTTP header field CoAP-Object-Security, whose value is:

- "" if the CoAP Object-Security option is empty, or
- the value of the CoAP Object-Security option (Section 6.1) in base64url encoding (Section 5 of [RFC4648]) without padding (see [RFC7515] Appendix C for implementation notes for this encoding).

Note that the value of the HTTP body is the CoAP payload, i.e. the OSCORE payload (Section 6.2).

The HTTP header field Content-Type is set to 'application/oscore' (see Section 12.5).

The resulting message is an OSCORE message that uses HTTP.

A receiving endpoint uses [RFC8075] to translate an HTTP message into a CoAP message, with the following addition. The HTTP message includes the CoAP-Object-Security header field, which is mapped to the CoAP Object-Security option in the following way. The CoAP Object-Security option value is:

- empty if the value of the HTTP CoAP-Object-Security header field is a single zero byte (0x00) represented by AA
- the value of the HTTP CoAP-Object-Security header field decoded from base64url (Section 5 of [RFC4648]) without padding (see [RFC7515] Appendix C for implementation notes for this decoding).

Note that the value of the CoAP payload is the HTTP body, i.e. the OSCORE payload (Section 6.2).

The resulting message is an OSCORE message that uses CoAP.

The endpoint can then verify the message according to the OSCORE processing and get a verified CoAP message. It can then translate the verified CoAP message into a verified HTTP message.

10.3. HTTP-to-CoAP Translation Proxy

Section 10.2 of [RFC7252] and [RFC8075] specify the behavior of an HTTP-to-CoAP proxy. As requested in Section 1 of [RFC8075], this section describes the HTTP mapping for the OSCORE protocol extension of CoAP.

The presence of the Object-Security option, both in requests and responses, is expressed in an HTTP header field named CoAP-Object-Security in the mapped request or response. The value of the field is:

- AA if the CoAP Object-Security option is empty, or
- the value of the CoAP Object-Security option (Section 6.1) in base64url encoding (Section 5 of [RFC4648]) without padding (see [RFC7515] Appendix C for implementation notes for this encoding).

The header field Content-Type 'application/oscore' (see Section 12.5) is used for OSCORE messages transported in HTTP. The CoAP Content-Format option is omitted for OSCORE messages transported in CoAP.

The value of the body is the OSCORE payload (Section 6.2).

Example:

Mapping and notation here is based on "Simple Form" (Section 5.4.1.1 of [RFC8075]).

[HTTP request -- Before client object security processing]

```
GET http://proxy.url/hc/?target_uri=coap://server.url/orders
 HTTP/1.1
```

[HTTP request -- HTTP Client to Proxy]

```
POST http://proxy.url/hc/?target_uri=coap://server.url/ HTTP/1.1
Content-Type: application/oscore
CoAP-Object-Security: CSU
Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

```
[CoAP request -- Proxy to CoAP Server]

  POST coap://server.url/
  Object-Security: 09 25
  Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

[CoAP request -- After server object security processing]

  GET coap://server.url/orders

[CoAP response -- Before server object security processing]

  2.05 Content
  Content-Format: 0
  Payload: Exterminate! Exterminate!

[CoAP response -- CoAP Server to Proxy]

  2.04 Changed
  Object-Security: [empty]
  Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[HTTP response -- Proxy to HTTP Client]

  HTTP/1.1 200 OK
  Content-Type: application/oscore
  CoAP-Object-Security: ""
  Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[HTTP response -- After client object security processing]

  HTTP/1.1 200 OK
  Content-Type: text/plain
  Body: Exterminate! Exterminate!
```

Note that the HTTP Status Code 200 in the next-to-last message is the mapping of CoAP Code 2.04 (Changed), whereas the HTTP Status Code 200 in the last message is the mapping of the CoAP Code 2.05 (Content), which was encrypted within the compressed COSE object carried in the Body of the HTTP response.

10.4. CoAP-to-HTTP Translation Proxy

Section 10.1 of [RFC7252] describes the behavior of a CoAP-to-HTTP proxy. RFC 8075 [RFC8075] does not cover this direction in any more detail and so an example instantiation of Section 10.1 of [RFC7252] is used below.

Example:

```
[CoAP request -- Before client object security processing]

  GET coap://proxy.url/
  Proxy-Uri=http://server.url/orders

[CoAP request -- CoAP Client to Proxy]

  POST coap://proxy.url/
  Proxy-Uri=http://server.url/
  Object-Security: 09 25
  Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

[HTTP request -- Proxy to HTTP Server]

  POST http://server.url/ HTTP/1.1
  Content-Type: application/oscore
  CoAP-Object-Security: CSU
  Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

[HTTP request -- After server object security processing]

  GET http://server.url/orders HTTP/1.1

[HTTP response -- Before server object security processing]

  HTTP/1.1 200 OK
  Content-Type: text/plain
  Body: Exterminate! Exterminate!

[HTTP response -- HTTP Server to Proxy]

  HTTP/1.1 200 OK
  Content-Type: application/oscore
  CoAP-Object-Security: ""
  Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response - Proxy to CoAP Client]

  2.04 Changed
  Object-Security: [empty]
  Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]
```

[CoAP response -- After client object security processing]

```
2.05 Content
Content-Format: 0
Payload: Exterminate! Exterminate!
```

Note that the HTTP Code 2.04 (Changed) in the next-to-last message is the mapping of HTTP Status Code 200, whereas the CoAP Code 2.05 (Content) in the last message is the value that was encrypted within the compressed COSE object carried in the Body of the HTTP response.

11. Security Considerations

11.1. End-to-end protection

In scenarios with intermediary nodes such as proxies or gateways, transport layer security such as (D)TLS only protects data hop-by-hop. As a consequence, the intermediary nodes can read and modify information. The trust model where all intermediary nodes are considered trustworthy is problematic, not only from a privacy perspective, but also from a security perspective, as the intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture brittle.

(D)TLS protects hop-by-hop the entire message. OSCORE protects end-to-end all information that is not required for proxy operations (see Section 4). (D)TLS and OSCORE can be combined, thereby enabling end-to-end security of the message payload, in combination with hop-by-hop protection of the entire message, during transport between endpoint and intermediary node. The CoAP messaging layer, including header fields such as Type and Message ID, as well as CoAP message fields Token and Token Length may be changed by a proxy and thus cannot be protected end-to-end. Error messages occurring during CoAP processing are protected end-to-end. Error messages occurring during OSCORE processing are not always possible to protect, e.g. if the receiving endpoint cannot locate the right security context. It may still be favorable to send an unprotected error message, e.g. to prevent extensive retransmissions, so unprotected error messages are allowed as specified. Similar to error messages, signaling messages are not always possible to protect as they may be intended for an intermediary. Hop-by-hop protection of signaling messages can be achieved with (D)TLS. Applications using unprotected error and signaling messages need to consider the threat that these messages may be spoofed.

11.2. Security Context Establishment

The use of COSE to protect messages as specified in this document requires an established security context. The method to establish the security context described in Section 3.2 is based on a common keying material in client and server, which may be obtained, e.g., by using the ACE framework [I-D.ietf-ace-oauth-authz]. An OSCORE profile of ACE is described in [I-D.ietf-ace-oscore-profile]. The key establishement procedure need to ensure same key is not installed twice, even in error situations.

11.3. Replay Protection

Most AEAD algorithms require a unique nonce for each message, for which the sender sequence numbers in the COSE message field 'Partial IV' is used. If the recipient accepts any sequence number larger than the one previously received, then the problem of sequence number synchronization is avoided. With reliable transport, it may be defined that only messages with sequence number which are equal to previous sequence number + 1 are accepted. The alternatives to sequence numbers have their issues: very constrained devices may not be able to support accurate time, or to generate and store large numbers of random 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.

11.4. Cryptographic Considerations

The maximum sender sequence number is dependent on the AEAD algorithm. The maximum sender sequence number is 2^40 - 1, or any algorithm specific lower limit, after which a new security context must be generated. The mechanism to build the nonce (Section 5.2) assumes that the nonce is at least 56 bits, and the Partial IV is at most 40 bits. The mandatory-to-implement AEAD algorithm AES-CCM-16-64-128 is selected for compatibility with CCM*.

The security level of a system with m Masters Keys of length k used together with Master Salts with entropy n is k + n - log2(m). Similarly, the security level of a system with m AEAD keys of length k used together with AEAD nonces of length n is k + n - log2(m). Security level here means that an attacker can recover one of the m keys with complexity 2^(k + n) / m. Protection against such attacks can be provided by increasing the size of the keys or the entropy of the Master Salt. The complexity of recovering a specific key is still 2^k (assuming the Master Salt/AEAD nonce is public) (see [MF00] for a overview). The Master Secret, Sender Key, and Recipient Key must be secret, the rest of the parameters may be public. The Master Secret must be uniformly random.

11.5. Message Segmentation

The Inner Block options enable the sender to split large messages into OSCORE-protected blocks such that the receiving endpoint can verify blocks before having received the complete message. The Outer Block options allow for arbitrary proxy fragmentation operations that cannot be verified by the endpoints, but can by policy be restricted in size since the Inner Block options allow for secure fragmentation of very large messages. A maximum message size (above which the sending endpoint fragments the message and the receiving endpoint discards the message, if complying to the policy) may be obtained as part of normal resource discovery.

11.6. Privacy Considerations

Privacy threats executed through intermediary nodes are considerably reduced by means of OSCORE. End-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations, provide mitigation against attacks on sensor and actuator communication, which may have a direct impact on the personal sphere.

The unprotected options (Figure 5) may reveal privacy sensitive information. In particular Uri-Host SHOULD NOT contain privacy sensitive information. CoAP headers sent in plaintext allow, for example, matching of CON and ACK (CoAP Message Identifier), matching of request and responses (Token) and traffic analysis. OSCORE does not provide protection for HTTP header fields which are not CoAP-mappable.

Unprotected error messages reveal information about the security state in the communication between the endpoints. Unprotected signalling messages reveal information about the reliable transport used on a leg of the path. Using the mechanisms described in Section 7.5 may reveal when a device goes through a reboot. This can be mitigated by the device storing the precise state of sender sequence number and replay window on a clean shutdown.

The length of message fields can reveal information about the message. Applications may use a padding scheme to protect against traffic analysis.

12. IANA Considerations

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

Note to IANA: Please note all occurrences of "TBD" in this specification should be assigned the same number.

12.1. COSE Header Parameters Registry

The 'kid context' parameter is added to the "COSE Header Parameters Registry":

- Name: kid context
- Label: TBD1 (Integer value between 1 and 255)
- Value Type: bstr
- Value Registry:
- Description: kid context
- Reference: Section 5.1 of this document

12.2. CoAP Option Numbers Registry

The Object-Security option is added to the CoAP Option Numbers registry:

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

12.3. CoAP Signaling Option Numbers Registry

The Object-Security option is added to the CoAP Signaling Option Numbers registry:

Applies to	Number	Name	Reference
7.xx (any)	TBD	Object-Security	[[this document]]

12.4. Header Field Registrations

The HTTP header field CoAP-Object-Security is added to the Message Headers registry:

Header Field Name	Protocol	Status	Reference
CoAP-Object-Security	http	standard	[[this document]]

12.5. Media Type Registrations

This section registers the 'application/oscore' media type in the "Media Types" registry.
These media types are used to indicate that the content is an OSCORE message.

Type name: application

Subtype name: oscore

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section of [[This document]].

Interoperability considerations: N/A

Published specification: [[This document]]

Applications that use this media type: IoT applications sending security content over HTTP(S) transports.

Fragment identifier considerations: N/A

Additional information:

* Deprecated alias names for this type: N/A

* Magic number(s): N/A

* File extension(s): N/A

* Macintosh file type code(s): N/A

Person & email address to contact for further information: iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander, goran.selander@ericsson.com

Change Controller: IESG

Provisional registration? No

12.6. CoAP Content-Formats Registry

TODO

13. References

13.1. Normative References

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

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

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

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

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

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

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

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

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

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

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

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

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

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

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

13.2. Informative References

[I-D.bormann-6lo-coap-802-15-ie]
Bormann, C., "Constrained Application Protocol (CoAP) over IEEE 802.15.4 Information Element for IETF", draft-bormann-6lo-coap-802-15-ie-00 (work in progress), April 2016.

[I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-03 (work in progress), July 2017.

[I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-05 (work in progress), March 2018.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-10 (work in progress), February 2018.

[I-D.ietf-ace-oscore-profile]
Seitz, L., Palombini, F., Gunnarsson, M., and G. Selander, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-01 (work in progress), March 2018.

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.

[I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-01 (work in progress), March 2018.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Secure group communication for CoAP", draft-ietf-core-oscore-groupcomm-01 (work in progress), March 2018.

[I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., Palombini, F., and C. Amsuess, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-04 (work in progress), March 2018.

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

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

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

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

[RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <https://www.rfc-editor.org/info/rfc7967>.

Appendix A. Scenario Examples

This section gives examples of OSCORE, targeting scenarios in Section 2.2.1.1 of [I-D.hartke-core-e2e-security-reqs]. The message exchanges are made, based on the assumption that there is a security context established between client and server. For simplicity, these examples only indicate the content of the messages without going into detail of the (compressed) COSE message format.

A.1. Secure Access to Sensor

This example illustrates a client requesting the alarm status from a server.

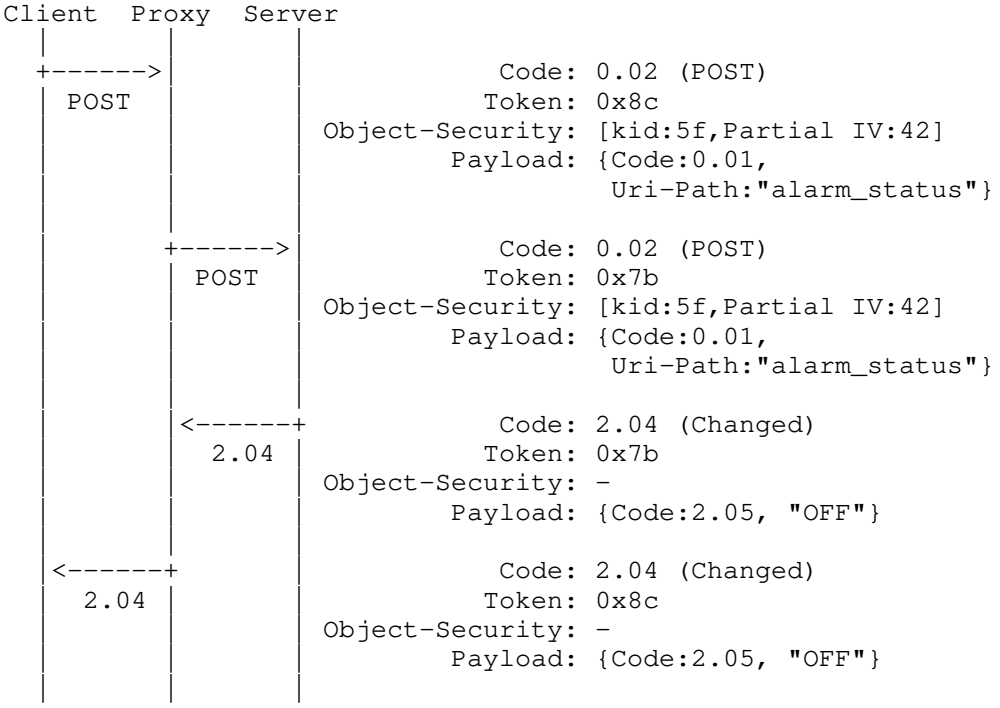


Figure 11: Secure Access to Sensor. Square brackets [...] indicate content of compressed COSE object. Curly brackets { ... } indicate encrypted data.

The request/response Codes are encrypted by OSCORE and only dummy Codes (POST/Changed) are visible in the header of the OSCORE message. The option Uri-Path ("alarm_status") and payload ("OFF") are encrypted.

The COSE header of the request contains an identifier (5f), indicating which security context was used to protect the message and a Partial IV (42).

The server verifies the request as specified in Section 8.2. The client verifies the response as specified in Section 8.4.

A.2. Secure Subscribe to Sensor

This example illustrates a client requesting subscription to a blood sugar measurement resource (GET /glucose), first receiving the value 220 mg/dl and then a second value 180 mg/dl.

Client Proxy Server

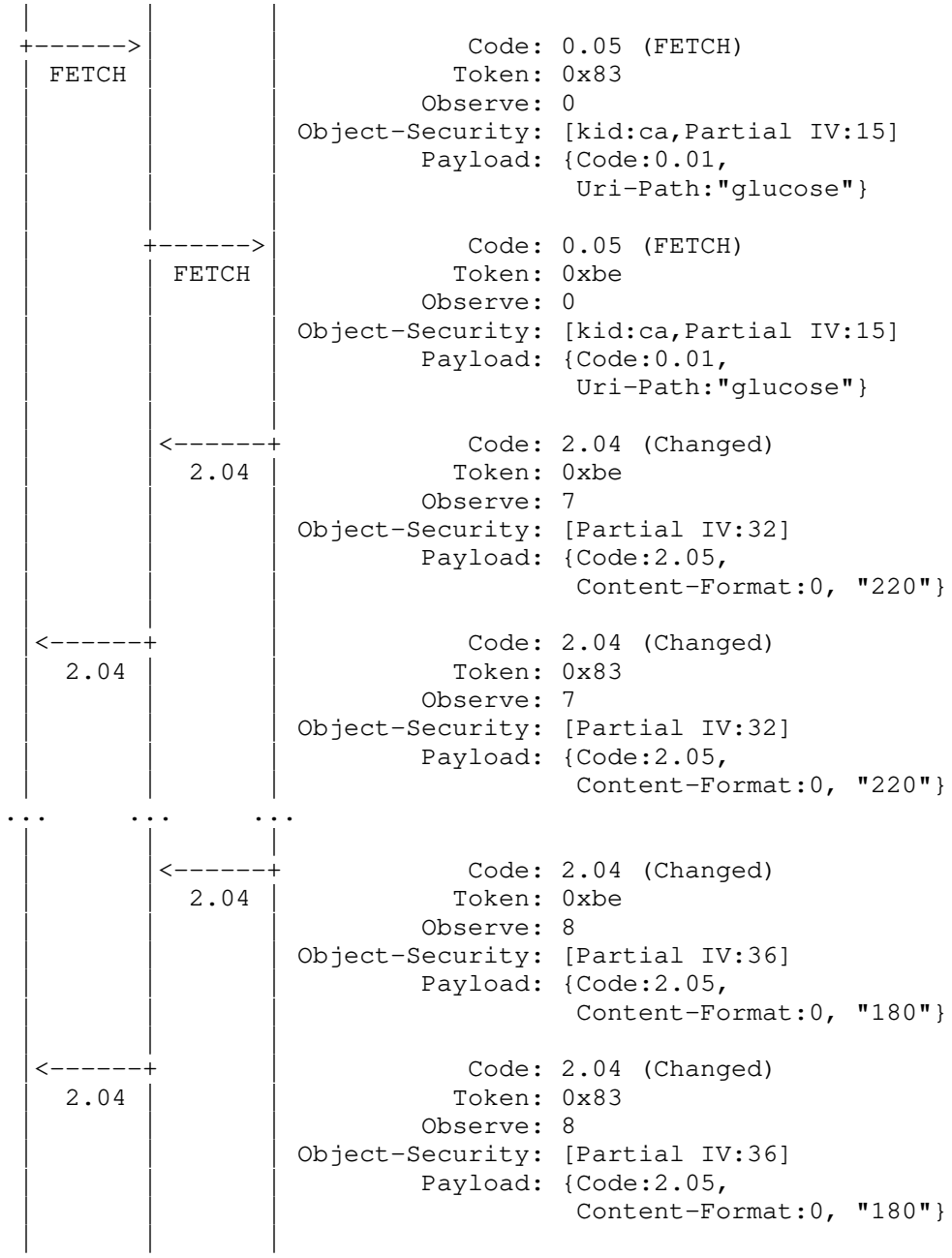


Figure 12: Secure Subscribe to Sensor. Square brackets [...] indicate content of compressed COSE object header. Curly brackets { ... } indicate encrypted data.

The request/response Codes are encrypted by OSCORE and only dummy Codes (FETCH/Changed) are visible in the header of the OSCORE message. The options Content-Format (0) and the payload ("220" and "180"), are encrypted.

The COSE header of the request contains an identifier (ca), indicating the security context used to protect the message and a Partial IV (15). The COSE headers of the responses contains Partial IVs (32 and 36).

The server verifies that the Partial IV has not been received before. The client verifies that the responses are bound to the request and that the Partial IVs are greater than any Partial IV previously received in a response bound to the request.

Appendix B. Deployment examples

OSCORE may be deployed in a variety of settings, a few examples are given in this section.

B.1. Master Secret Used Once

For settings where the Master Secret is only used during deployment, the uniqueness of AEAD nonce may be assured by persistent storage of the security context as described in this specification (see Section 7.5). For many IoT deployments, a 128 bit uniformly random Master Key is sufficient for encrypting all data exchanged with the IoT device throughout its lifetime.

B.2. Master Secret Used Multiple Times

In cases where the Master Secret needs to be used to derive multiple security contexts, e.g. due to recommissioning or where the security context is not persistently stored, a stochastically unique Master Salt prevents the reuse of AEAD nonce and key. The Master Salt may be transported between client and server in the kid context parameter (see Section 5.1) of the request.

In this section we give an example of a procedure which may be implemented in client and server to establish the OSCORE security context based on pre-established input parameters (see Section 3.2) except for the Master Salt which is transported in kid context.

1. In order to establish a security context with a server for the first time, or a new security context replacing an old security context, the client generates a (pseudo-)random uniformly distributed 64-bit Master Salt and derives the security context as specified in Section 3.2. The client protects a request with

the new Sender Context and sends the message with kid context set to the Master Salt.

2. The server, receiving an OSCORE request with a non-empty kid context derives the new security context using the received kid context as Master Salt. The server processes the request as specified in this document using the new Recipient Context. If the processing of the request completes without error, the server responds with an Echo option as specified in [I-D.ietf-core-echo-request-tag]. The response is protected with the new Sender Context.

3. The client, receiving a response with an Echo option to a request which used a new security context, verifies the response using the new Recipient Context, and if valid repeats the request with the Echo option (see [I-D.ietf-core-echo-request-tag]) using the new Sender Context. Subsequent message exchanges (unless superseded) are processed using the new security context without including the Master Salt in the kid context.

4. The server, receiving a request with a kid context and a valid Echo option (see [I-D.ietf-core-echo-request-tag]), repeats the processing described in step 2. If it completes without error, then the new security context is established, and the request is valid. If the server already had an old security context with this client that is now replaced by the new security context.

If the server receives a request without kid context from a client with which no security context is established, then the server responds with a 4.01 Unauthorized error message with diagnostic payload containing the string "Security context not found". This could be the result of the server having lost its security context or that a new security context has not been successfully established, which may be a trigger for the client to run this procedure.

B.3. Client Aliveness

The use of a single OSCORE request and response enables the client to verify that the server's identity and aliveness through actual communications. While a verified OSCORE request enables the server to verify the identity of the entity who generated the message, it does not verify that the client is currently involved in the communication, since the message may be a delayed delivery of a previously generated request which now reaches the server. To verify the aliveness of the client the server may initiate an OSCORE protected message exchange with the client, e.g. by switching the roles of client and server as described in Section 3.1, or by using

the Echo option in the response to a request from the client [I-D.ietf-core-echo-request-tag].

Appendix C. Test Vectors

This appendix includes the test vectors for different examples of CoAP messages using OSCORE.

C.1. Test Vector 1: Key Derivation with Master Salt

Given a set of inputs, OSCORE defines how to set up the Security Context in both the client and the server. The default values are used for AEAD Algorithm and KDF.

C.1.1. Client

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Master Salt: 0x9e7ca92223786340 (8 bytes)
- Sender ID: 0x (0 byte)
- Recipient ID: 0x01 (1 byte)

From the previous parameters,

- info (for Sender Key): 0x84400A634b657910 (8 bytes)
- info (for Recipient Key): 0x8441010A634b657910 (9 bytes)
- info (for Common IV): 0x84400a6249560d (7 bytes)

Outputs:

- Sender Key: 0x7230aab3b549d94c9224aacc744e93ab (16 bytes)
- Recipient Key: 0xe534a26a64aa3982e988e31f1e401e65 (16 bytes)
- Common IV: 0x01727733ab49ead385b18f7d91 (13 bytes)

C.1.2. Server

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)

- Master Salt: 0x9e7ca92223786340 (64 bytes)
- Sender ID: 0x01 (1 byte)
- Recipient ID: 0x (0 byte)

From the previous parameters,

- info (for Sender Key): 0x8441010A634b657910 (9 bytes)
- info (for Recipient Key): 0x84400A634b657910 (8 bytes)
- info (for Common IV): 0x84400a6249560d (7 bytes)

Outputs:

- Sender Key: 0xe534a26a64aa3982e988e31f1e401e65 (16 bytes)
- Recipient Key: 0x7230aab3b549d94c9224aacc744e93ab (16 bytes)
- Common IV: 0x01727733ab49ead385b18f7d91 (13 bytes)

C.2. Test Vector 2: Key Derivation without Master Salt

Given a set of inputs, OSCORE defines how to set up the Security Context in both the client and the server. The default values are used for AEAD Algorithm, KDF, and Master Salt.

C.2.1. Client

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Sender ID: 0x00 (1 byte)
- Recipient ID: 0x01 (1 byte)

From the previous parameters,

- info (for Sender Key): 0x8441000A634b657910 (9 bytes)
- info (for Recipient Key): 0x8441010A634b657910 (9 bytes)
- info (for Common IV): 0x84400a6249560d (7 bytes)

Outputs:

- Sender Key: 0xf8f3b887436285ed5a66f6026ac2cdc1 (16 bytes)
- Recipient Key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)
- Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

C.2.2. Server

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Sender ID: 0x01 (1 byte)
- Recipient ID: 0x00 (1 byte)

From the previous parameters,

- info (for Sender Key): 0x8441010A634b657910 (9 bytes)
- info (for Recipient Key): 0x8441000A634b657910 (9 bytes)
- info (for Common IV): 0x84400a6249560d (7 bytes)

Outputs:

- Sender Key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)
- Recipient Key: 0xf8f3b887436285ed5a66f6026ac2cdc1 (16 bytes)
- Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

C.3. Test Vector 3: OSCORE Request, Client

This section contains a test vector for a OSCORE protected CoAP GET request using the security context derived in Appendix C.1. The unprotected request only contains the Uri-Path option.

Unprotected CoAP request:
0x440149c60000f2a7396c6f63616c686f737483747631 (22 bytes)

Common Context:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)
- Key Derivation Function: HKDF SHA-256
- Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

Sender Context:

- Sender ID: 0x00 (1 byte)
- Sender Key: 0xf8f3b887436285ed5a66f6026ac2cdc1 (16 bytes)
- Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- Partial IV: 0x14 (1 byte)
- kid: 0x00 (1 byte)
- external_aad: 0x8501810a4100411440 (9 bytes)
- AAD: 0x8368456e63727970743040498501810a4100411440 (21 bytes)
- plaintext: 0x01b3747631 (5 bytes)
- encryption key: 0xf8f3b887436285ed5a66f6026ac2cdc1 (16 bytes)
- nonce: 0xd0a1949aa253278f34c528d2d8 (13 bytes)

From the previous parameter, the following is derived:

- Object-Security value: 0x091400 (3 bytes)
- ciphertext: 0x55b3710d47c611cd3924838a44 (13 bytes)

From there:

- Protected CoAP request (OSCORE message): 0x44026dd30000acc5396c6f6 3616c686f7374d305091400ff55b3710d47c611cd3924838a44 (37 bytes)

C.4. Test Vector 4: OSCORE Request, Client

This section contains a test vector for a OSCORE protected CoAP GET request using the security context derived in Appendix C.2. The unprotected request only contains the Uri-Path option.

Unprotected CoAP request:
0x440149c60000f2a7396c6f63616c686f737483747631 (22 bytes)

Common Context:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)

- Key Derivation Function: HKDF SHA-256
- Common IV: 0x01727733ab49ead385b18f7d91 (13 bytes)

Sender Context:

- Sender ID: 0x (0 bytes)
- Sender Key: 0x7230aab3b549d94c9224aacc744e93ab (16 bytes)
- Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- Partial IV: 0x14 (1 byte)
- kid: 0x (0 byte)
- external_aad: 0x8501810a40411440 (8 bytes)
- AAD: 0x8368456e6372797074304048850181 0a40411440 (20 bytes)
- plaintext: 0x01b3747631 (5 bytes)
- encryption key: 0x7230aab3b549d94c9224aacc744e93ab (16 bytes)
- nonce: 0x01727733ab49ead385b18f7d85 (13 bytes)

From the previous parameter, the following is derived:

- Object-Security value: 0x0914 (2 bytes)
- ciphertext: 0x6be9214aad448260ff1be1f594 (13 bytes)

From there:

- Protected CoAP request (OSCORE message): 0x44023bfc000066ef396c6f6 3616c686f7374d2050914ff6be9214aad448260ff1be1f594 (36 bytes)

C.5. Test Vector 5: OSCORE Response, Server

This section contains a test vector for a OSCORE protected 2.05 Content response to the request in Appendix C.3. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a kid nor a Partial IV.

Unprotected CoAP response:
0x644549c60000f2a7ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)
- Key Derivation Function: HKDF SHA-256
- Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

Sender Context:

- Sender ID: 0x01 (1 byte)
- Sender Key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)
- Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- external_aad: 0x8501810a4100411440 (9 bytes)
- AAD: 0x8368456e6372797074304049850181 0a4100411440 (21 bytes)
- plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- encryption key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)
- nonce: 0xd0a1949aa253278f34c528d2d8 (13 bytes)

From the previous parameter, the following is derived:

- Object-Security value: 0x (0 bytes)
- ciphertext: e4e8c28c41c8f31ca56eec24f6c71d94eacbcdffdc6d (22 bytes)

From there:

- Protected CoAP response (OSCORE message): 0x64446dd30000acc5d008ff e4e8c28c41c8f31ca56eec24f6c71d94eacbcdffdc6d (33 bytes)

C.6. Test Vector 6: OSCORE Response with Partial IV, Server

This section contains a test vector for a OSCORE protected 2.05 Content response to the request in Appendix C.3. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a kid, but contains a Partial IV.

Unprotected CoAP response:
0x644549c60000f2a7ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)
- Key Derivation Function: HKDF SHA-256
- Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

Sender Context:

- Sender ID: 0x01 (1 byte)
- Sender Key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)
- Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- Partial IV: 0x00 (1 byte)
- external_aad: 0x8501810a4100411440 (9 bytes)
- AAD: 0x8368456e6372797074304049850181 0a4100411440 (21 bytes)
- plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- encryption key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)
- nonce: 0xd0a1949aa253278e34c528d2cc (13 bytes)

From the previous parameter, the following is derived:

- Object-Security value: 0x0100 (2 bytes)
- ciphertext: 0xa7e3ca27f221f453c0ba68c350bf652ea096b328a1bf (22 bytes)

From there:

- Protected CoAP response (OSCORE message): 0x64442b130000b29ed20801 00ffa7e3ca27f221f453c0ba68c350bf652ea096b328a1bf (35 bytes)

Appendix D. Overview of Security Properties

D.1. Supporting Proxy Operations

CoAP is designed to work with intermediaries reading and/or changing CoAP message fields and performing supporting operations in constrained environments, e.g. forwarding and cross-protocol translations.

Securing CoAP on transport layer protects the entire message between the endpoints in which case CoAP proxy operations are not possible. In order to enable proxy operations, security on transport layer needs to be terminated at the proxy in which case the CoAP message in its entirety is unprotected in the proxy.

Requirements for CoAP end-to-end security are specified in [I-D.hartke-core-e2e-security-reqs]. The client and server are assumed to trust each other, but proxies and gateways are only trusted to perform its intended operations. Forwarding is specified in Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs]. HTTP-CoAP translation is specified in [RFC8075]. Intermediaries translating between different transport layers are intended to perform just that.

By working at the CoAP layer, OSCORE enables different CoAP message fields to be protected differently, which allows message fields required for proxy operations to be available to the proxy while message fields intended for the other endpoint remain protected. In the remainder of this section we analyze how OSCORE protects the protected message fields and the consequences of message fields intended for proxy operation being unprotected.

D.2. Protected Message Fields

Protected message fields are included in the Plaintext (Section 5.3) and the Additional Authenticated Data (Section 5.4) of the COSE_Encrypt0 object using an AEAD algorithm.

OSCORE depends on a pre-established strong Master Secret which can be used to derive keys, and a construction for making (key, nonce) pairs unique (Appendix D.3). Assuming this is true, and the keys are used for no more data than indicated in Section 7.2, OSCORE should provide the following guarantees:

- Confidentiality: An attacker should not be able to determine the plaintext contents of a given OSCORE message or determine that different plaintexts are related (Section 5.3).

- Integrity: An attacker should not be able to craft a new OSCORE message with protected message fields different from an existing OSCORE message which will be accepted by the receiver.

- Request-response binding: An attacker should not be able to make a client match a response to the wrong request.

- Non-replayability: An attacker should not be able to cause the receiver to accept a message which it has already accepted.

Informally, OSCORE provides these properties by AEAD-protecting the plaintext with a strong key and uniqueness of (key, nonce) pairs. AEAD encryption [RFC5116] provides confidentiality and integrity for the data. Response-request binding is provided by including the kid and Partial IV of the request in the AAD of the response. Non-replayability of requests and notifications is provided by using unique (key, nonce) pairs and a replay protection mechanism (application dependent, see Section 7.4).

OSCORE is susceptible to a variety of traffic analysis attacks based on observing the length and timing of encrypted packets. OSCORE does not provide any specific defenses against this form of attack but the application may use a padding mechanism to prevent an attacker from directly determine the length of the padding. However, information about padding may still be revealed by side-channel attacks observing differences in timing.

D.3. Uniqueness of (key, nonce)

In this section we show (key, nonce) pairs are not reused in the encryption of OSCORE messages.

Fix a Security Context complying with the requirements Section 3.3) and an endpoint. Endpoints may alternate between Client and Server roles, but each endpoint encrypts with the Sender Key of its Sender Context. Sender Keys are (stochastically) unique since they are derived with HKDF from unique Sender IDs, so messages encrypted by different endpoints use different keys. It remains to prove that the nonces used by the fixed endpoint are unique.

Since the Common IV is fixed, the nonces are determined by a Partial IV (PIV) and the Sender ID of the endpoint generating that Partial IV (ID_PIV), and are unique for different (ID_PIV, PIV) pairs (Section 5.2).

For requests and notifications (GET Observe responses):

- ID_PIV = Sender ID of the encrypting endpoint

- PIV = current Partial IV of the encrypting endpoint

Since the encrypting endpoint steps the Partial IV for each use, the nonces used in requests and notifications are all unique as long as the number of encrypted messages are kept within the required range (Section 7.2).

For responses to requests:

- ID_PIV = Sender ID of the endpoint generating the request

- PIV = Partial IV of the request

Since the request has been verified using the Recipient Context, ID_PIV is the Sender ID of another endpoint and is thus different from the Sender ID of the encrypting endpoint. Therefore the nonces used in responses are different compared to nonces in requests and notifications. Since the Partial IV of the request is verified for replay (Section 7.4), PIV is unique for responses and so are nonces used in responses.

Note that the argument does not depend on if the nonce in the first response to GET Observe is generated as a notification or as a response to a request. In the former case the Partial IV of the encrypting endpoint is stepped. In the latter case, the nonce is in the the requesting endpoint's subset of nonces and would otherwise not be used by the encrypting endpoint.

The argumentation also holds for group communication as specified in [RFC7390] although Observe is not used for that setting (see [I-D.ietf-core-oscore-groupcomm]).

D.4. Unprotected Message Fields

This section lists and discusses issues with unprotected CoAP message fields.

D.4.1. CoAP Header Fields

- Version

The CoAP version will be in plaintext. A change of this parameter is potentially a denial of service attack. Currently there is only one CoAP version defined. Future versions of CoAP need to analyse attacks to OSCORE protected messages due to an adversary changing the CoAP version.

- Token/Token Length

The Token field is a client-local identifier for differentiating between concurrent requests. Change of Token is a denial of service attack, since the client may not be able to identify the request or verify integrity of the response, which depends on the request.

- Type/Message ID

These fields reveal information about the UDP transport binding. CoAP proxies are allowed to change Type and Message ID. These message fields are not present in CoAP over TCP, and does not impact the request/response message. A change of these fields is a denial of service attack similar to changing UDP header fields.

- Length

This field reveal information about the TCP transport binding. These message fields are not present in CoAP over UDP, and does not impact the request/response message. A change of Length is a denial of service attack similar to changing TCP header fields.

D.4.2. CoAP Options

- Max-Age

The Outer Max-Age is used to avoid unnecessary caching of OSCORE error responses. Changing this value is a potential denial of service attack.

- Proxy-Uri/Proxy-Scheme/Uri-Host/Uri-Port

With OSCORE, the Proxy-Uri option does not contain the Uri-Path/Uri-Query parts of the URI. Proxy-Uri/Proxy-Scheme/Uri-Host/Uri-Port cannot be integrity protected since they are allowed to be changed by a forward proxy.

- Observe

The Outer Observe option is intended for an OSCORE-unaware proxy to support forwarding of Observe messages. Changing this option may lead to notifications not being forwarded.

- Block1/Block2/Size1/Size2

The Outer Block options enables fragmentation of OSCORE messages in addition to segmentation performed by the Inner Block options. Manipulating these options is a potential denial of service attack, e.g. injection of alleged Block fragments up to the MAX_UNFRAGMENTED_SIZE, at which the message will be dropped.

- No-Response

The Outer No-Response option is used to support proxy functionality, specifically to avoid error transmissions from proxies to clients, and to avoid bandwidth reduction to servers by proxies applying congestion control when not receiving responses. Changing this option is a potential denial of service attack.

- Object-Security

The Object-Security option contains information about the compressed COSE header. A change of this field may result in not being able to verify the OSCORE message.

Appendix E. CDDL Summary

Data structure definitions in the present specification employ the CDDL language for conciseness and precision. CDDL is defined in [I-D.ietf-cbor-cddl], which at the time of writing this appendix is in the process of completion. As the document is not yet available for a normative reference, the present appendix defines the small subset of CDDL that is being used in the present specification.

Within the subset being used here, a CDDL rule is of the form "name = type", where "name" is the name given to the "type". A "type" can be one of:

- a reference to another named type, by giving its name. The predefined named types used in the present specification are: "uint", an unsigned integer (as represented in CBOR by major type 0); "int", an unsigned or negative integer (as represented in CBOR by major type 0 or 1); "bstr", a byte string (as represented in CBOR by major type 2); "tstr", a text string (as represented in CBOR by major type 3);

- a choice between two types, by giving both types separated by a "/";

- an array type (as represented in CBOR by major type 4), where the sequence of elements of the array is described by giving a sequence of entries separated by commas ",", and this sequence is enclosed by square brackets "[" and "]". Arrays described by an array description contain elements that correspond one-to-one to the sequence of entries given. Each entry of an array description is of the form "name : type", where "name" is the name given to the entry and "type" is the type of the array element corresponding to this entry.

Acknowledgments

The following individuals provided input to this document: Christian Amsuess, Tobias Andersson, Carsten Bormann, Joakim Brorsson, Esko Dijk, Thomas Fossati, Martin Gunnarsson, Klaus Hartke, Jim Schaad, Peter van der Stok, Dave Thaler, Marco Tiloca, and Malisa Vucinic.

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova.

Authors' Addresses

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com

Ludwig Seitz
RISE SICS

Email: ludwig.seitz@ri.se

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

M. Tiloca
RISE SICS AB
G. Selander
F. Palombini
Ericsson AB
J. Park
Universitaet Duisburg-Essen

March 05, 2018

Secure group communication for CoAP
draft-ietf-core-oscore-groupcomm-01

Abstract

This document describes a mode for protecting group communication over the Constrained Application Protocol (CoAP). The proposed mode relies on Object Security for Constrained RESTful Environments (OSCORE) and the CBOR Object Signing and Encryption (COSE) format. In particular, it is defined how OSCORE should be used in a group communication setting, while fulfilling the same security requirements for request messages and related response messages. Source authentication of all messages exchanged within the group is ensured, by means of digital signatures produced through private keys of sender endpoints and embedded in the protected CoAP messages.

Table of Contents

1. Introduction 3
1.1. Terminology 3
2. OSCORE Security Context 5
2.1. Management of Group Keying Material 7
3. The COSE Object 8
3.1. Example: Request 10
3.2. Example: Response 10
4. Message Processing 11
4.1. Protecting the Request 11
4.2. Verifying the Request 12
4.3. Protecting the Response 12
4.4. Verifying the Response 12
5. Synchronization of Sequence Numbers 13
6. Responsibilities of the Group Manager 13
7. Security Considerations 15
7.1. Group-level Security 15
8. IANA Considerations 15
9. Acknowledgments 15
10. References 15
10.1. Normative References 15
10.2. Informative References 16
Appendix A. Assumptions and Security Objectives 19
A.1. Assumptions 19
A.2. Security Objectives 20
Appendix B. List of Use Cases 21
Appendix C. Example of Group Identifier Format 23
Appendix D. Set-up of New Endpoints 24
D.1. Join Process 24
D.2. Provisioning and Retrieval of Public Keys 27
D.3. Group Joining Based on the ACE Framework 29
Appendix E. Examples of Synchronization Approaches 29
E.1. Best-Effort Synchronization 30
E.2. Baseline Synchronization 30
E.3. Challenge-Response Synchronization 30
Appendix F. No Verification of Signatures 32
Authors' Addresses 32

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol specifically designed for constrained devices and networks [RFC7228].

Group communication for CoAP [RFC7390] addresses use cases where deployed devices benefit from a group communication model, for example to reduce latencies and improve performance. Use cases include lighting control, integrated building control, software and firmware updates, parameter and configuration updates, commissioning of constrained networks, and emergency multicast (see Appendix B). Furthermore, [RFC7390] recognizes the importance to introduce a secure mode for CoAP group communication. This specification defines such a mode.

Object Security for Constrained RESTful Environments (OSCORE)[I-D.ietf-core-object-security] describes a security protocol based on the exchange of protected CoAP messages. OSCORE builds on CBOR Object Signing and Encryption (COSE) [RFC8152] and provides end-to-end encryption, integrity, and replay protection between a sending endpoint and a receiving endpoint possibly involving intermediary endpoints. To this end, a CoAP message is protected by including its payload (if any), certain options, and header fields in a COSE object, which finally replaces the authenticated and encrypted fields in the protected message.

This document describes group OSCORE, providing end-to-end security of CoAP messages exchanged between members of a group. In particular, the described approach defines how OSCORE should be used in a group communication setting, so that end-to-end security is assured by using the same security method. That is, end-to-end security is assured for multicast CoAP requests sent by multicaster endpoints to the group and for related CoAP responses sent as reply by multiple listener endpoints. Group OSCORE provides source authentication of all CoAP messages exchanged within the group, by means of digital signatures produced through private keys of sender devices and embedded in the protected CoAP messages. As in OSCORE, it is still possible to simultaneously rely on DTLS to protect hop-by-hop communication between a multicaster endpoint and a proxy (and vice versa), and between a proxy and a listener endpoint (and vice versa).

1.1. Terminology

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

14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252] including "endpoint", "sender" and "recipient"; group communication for CoAP [RFC7390]; COSE and counter signatures [RFC8152].

Readers are also expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE, such as "Security Context", "Master Secret" and "Master Salt", defined in [I-D.ietf-core-object-security].

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

This document refers also to the following terminology.

- Keying material: data that is necessary to establish and maintain secure communication among endpoints. This includes, for instance, keys and IVs [RFC4949].

- Group: a set of endpoints that share group keying material and parameters (Common Context of the group's Security Context, see Section 2). That is, the term group used in this specification refers to a "security group", not to be confused with network/ multicast groups or application groups.

- Group Manager (GM): entity responsible for a set of OSCORE groups. Each endpoint in a group securely communicates with the respective GM, which is not required to be an actual group member and to take part in the group communication. The full list of responsibilities of the Group Manager is provided in Section 6.

- Multicaster: member of a group that sends multicast CoAP request messages intended for all members of the group. In a 1-to-N communication model, only a single multicaster transmits data to the group; in an M-to-N communication model (where M and N do not necessarily have the same value), M group members are multicasters. According to [RFC7390], any possible proxy entity is supposed to know about the multicasters in the group and to not perform aggregation of response messages. Also, every multicaster expects and is able to handle multiple response messages associated to a given multicast request message that it has previously sent to the group.

- Listener: member of a group that receives multicast CoAP request messages when listening to the multicast IP address associated to

the group. A listener may reply back, by sending a response message to the multicaster which has sent the request message.

- Pure listener: member of a group that is configured as listener and never replies back to multicasters after receiving request messages.

- Group ID: group identifier assigned to the group. Group IDs are unique within the set of groups of a same Group Manager.

- Endpoint ID: Sender ID of the endpoint, as defined in [I-D.ietf-core-object-security]. An Endpoint ID is provided to an endpoint upon joining a group, is valid only within that group, and is unique within the same group. Endpoints which are configured only as pure listeners do not have an Endpoint ID.

- Group request: multicast CoAP request message sent by a multicaster in the group to all listeners in the group through multicast IP, unless otherwise specified.

- Source authentication: evidence that a received message in the group originated from a specifically identified group member. This also provides assurances that the message was not tampered with by a different group member or by a non-group member.

2. OSCORE Security Context

To support group communication secured with OSCORE, each endpoint registered as member of a group maintains a Security Context as defined in Section 3 of [I-D.ietf-core-object-security]. In particular, each endpoint in a group stores:

1. one Common Context, shared by all the endpoints in the group. All the endpoints in the group agree on the same COSE AEAD algorithm. In addition to what is defined in Section 3 of [I-D.ietf-core-object-security], the Common Context includes the following information.

   * Group Identifier (Gid). Variable length byte string identifying the Security Context. A Gid MUST have a random component and be long enough, in order to achieve a negligible probability of collisions between Group Identifiers from different Group Managers. A Group ID is used i) alone or together with other parameters, such as the multicast IP address of the group, to retrieve the OSCORE Security Context of the associated group (see Section 4); and ii) as OSCORE Master Salt (see Section 3.1 of [I-D.ietf-core-object-security]). The choice of the Gid for a

given group's Security Context is application specific. It is the role of the application to specify how to handle possible collisions. An example of specific formatting of the Group Identifier that would follow this specification is given in Appendix C.

* Counter Signature Algorithm. Value identifying the algorithm used for source authenticating messages sent within the group, by means of a counter signature (see Section 4.5 of [RFC8152]). Its value is immutable once the Common Context is established. All the endpoints in the group agree on the same counter signature algorithm. The list of supported signature algorithms is part of the group communication policy and MUST include the EdDSA signature algorithm ed25519 [RFC8032].

2. one Sender Context, unless the endpoint is configured exclusively as pure listener. The Sender Context is used to secure outgoing group messages and is initialized according to Section 3 of [I-D.ietf-core-object-security], once the endpoint has joined the group. In practice, the symmetric keying material in the Sender Context of the sender endpoint is shared with all the recipient endpoints that have received group messages from that same sender endpoint. Besides, in addition to what is defined in [I-D.ietf-core-object-security], the Sender Context stores also the endpoint's public-private key pair.

3. one Recipient Context for each distinct endpoint from which group messages are received, used to process such incoming messages. The recipient endpoint creates a new Recipient Context upon receiving an incoming message from another endpoint in the group for the first time (see Section 4.2 and Section 4.4). In practice, the symmetric keying material in a given Recipient Context of the recipient endpoint is shared with the associated sender endpoint from which group messages are received. Besides, in addition to what is defined in [I-D.ietf-core-object-security], each Recipient Context stores also the public key of the associated other endpoint from which group messages are received.

The table in Figure 1 overviews the new information included in the OSCORE Security Context, with respect to what defined in Section 3 of [I-D.ietf-core-object-security].

Context portion	New information
Common Context	Group Identifier (Gid)
Common Context	Counter signature algorithm
Sender Context	Endpoint's private key
Sender Context	Endpoint's public key
Each Recipient Context	Public key of the associated other endpoint

Figure 1: Additions to the OSCORE Security Context

Upon receiving a secure CoAP message, a recipient endpoint relies on the sender endpoint's public key, in order to verify the counter signature conveyed in the COSE Object.

If not already stored in the Recipient Context associated to the sender endpoint, the recipient endpoint retrieves the public key from a trusted key repository. In such a case, the correct binding between the sender endpoint and the retrieved public key must be assured, for instance by means of public key certificates. Further discussion about how public keys can be handled and retrieved in the group is provided in Appendix D.2.

The Sender Key/IV stored in the Sender Context and the Recipient Keys/IVs stored in the Recipient Contexts are derived according to the same scheme defined in Section 3.2 of [I-D.ietf-core-object-security].

2.1. Management of Group Keying Material

The approach described in this specification should take into account the risk of compromise of group members. In particular, the adoption of key management schemes for secure revocation and renewal of Security Contexts and group keying material should be considered.

Consistently with the security assumptions in Appendix A.1, it is RECOMMENDED to adopt a group key management scheme, and securely distribute a new value for the Master Secret parameter of the group's Security Context, before a new joining endpoint is added to the group or after a currently present endpoint leaves the group. This is

necessary in order to preserve backward security and forward security in the group.

In particular, a new Group Identifier (Gid) for that group and a new value for the Master Secret parameter must also be distributed. An example of Group Identifier format supporting this operation is provided in Appendix C. Then, each group member re-derives the keying material stored in its own Sender Context and Recipient Contexts as described in Section 2, using the updated Group Identifier.

Especially in dynamic, large-scale, groups where endpoints can join and leave at any time, it is important that the considered group key management scheme is efficient and highly scalable with the group size, in order to limit the impact on performance due to the Security Context and keying material update.

3. The COSE Object

When creating a protected CoAP message, an endpoint in the group computes the COSE object using the untagged COSE_Encrypt0 structure [RFC8152] as defined in Section 5 of [I-D.ietf-core-object-security], with the following modifications.

- The value of the "kid" parameter in the "unprotected" field of response messagess SHALL be set to the Endpoint ID of the endpoint transmitting the message, i.e. the Sender ID.

- The "unprotected" field of the "Headers" field SHALL additionally include the following parameter:

  * CounterSignature0 : its value is set to the counter signature of the COSE object, computed by the endpoint by means of its own private key as described in Section 4.5 of [RFC8152]. The presence of this parameter is explicitly signaled, by using the reserved sixth least significant bit of the first byte of flag bits in the value of the Object-Security option (see Section 6.1 of [I-D.ietf-core-object-security]).

- The Additional Authenticated Data (AAD) considered to compute the COSE object is extended, by adding the countersignature algorithm used to protect group messages. In particular, the "external_aad" defined in Section 5.4 of [I-D.ietf-core-object-security] SHALL also include "alg_countersign", which contains the Counter Signature Algorithm from the Common Context (see Section 2).

```
external_aad = [
   oscore_version : uint,
   [alg_aead : int / tstr , alg_countersign : int / tstr],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr
]
```

o The OSCORE compression defined in Section 6 of [I-D.ietf-core-object-security] is used, with the following additions for the encoding of the Object-Security option.

  * The fourth least significant bit of the first byte of flag bits SHALL be set to 1, to indicate the presence of the "kid" parameter for both group requests and responses.

  * The fifth least significant bit of the first byte of flag bits MUST be set to 1 for group requests, to indicate the presence of the kid context in the OSCORE payload. The kid context flag MAY be set to 1 for responses.

  * The sixth least significant bit of the first byte of flag bits is originally marked as reserved in [I-D.ietf-core-object-security] and its usage is defined in this specification. This bit is set to 1 if the "CounterSignature0" parameter is present, or to 0 otherwise. In order to ensure source authentication of group messages as described in this specification, this bit SHALL be set to 1.

  * The 'kid context' value encodes the Group Identifier value (Gid) of the group's Security Context.

  * The following q bytes (q given by the Counter Signature Algorithm specified in the Security Context) encode the value of the "CounterSignature0" parameter including the counter signature of the COSE object.

  * The remaining bytes in the Object-Security value encode the value of the "kid" parameter, which is always present both in group requests and in responses.

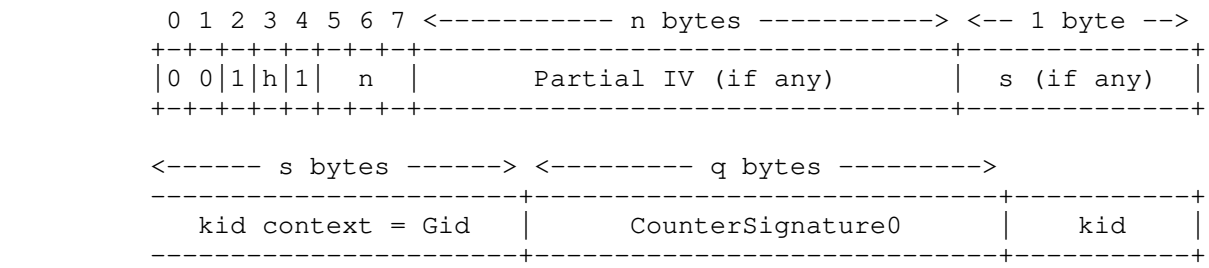


Figure 2: Object-Security Value

3.1. Example: Request

Request with kid = 0x25, Partial IV = 5 and kid context = 0x44616c, assuming the label for the new kid context defined in [I-D.ietf-core-object-security] has value 10. COUNTERSIGN is the CounterSignature0 byte string as described in Section 3 and is 64 bytes long in this example. The ciphertext in this example is 14 bytes long.

Before compression (96 bytes):

```
[
h'',
{ 4:h'25', 6:h'05', 10:h'44616c', 9:COUNTERSIGN },
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (85 bytes):

Flag byte: 0b00111001 = 0x39

Option Value: 39 05 03 44 61 6c COUNTERSIGN 25 (7 bytes + size of COUNTERSIGN)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

3.2. Example: Response

Response with kid = 0x52. COUNTERSIGN is the CounterSignature0 byte string as described in Section 3 and is 64 bytes long in this example. The ciphertext in this example is 14 bytes long.

Before compression (88 bytes):

```
[
h'',
{ 4:h'52', 9:COUNTERSIGN },
h'60b035059d9ef5667c5a0710823b'
]
```

After compression (80 bytes):

Flag byte: 0b00101000 = 0x28

Option Value: 28 COUNTERSIGN 52 (2 bytes + size of COUNTERSIGN)

Payload: 60 b0 35 05 9d 9e f5 66 7c 5a 07 10 82 3b (14 bytes)

4. Message Processing

Each request message and response message is protected and processed as specified in [I-D.ietf-core-object-security], with the modifications described in the following sections. The following security objectives are fulfilled, as further discussed in Appendix A.2: data replay protection, group-level data confidentiality, source authentication, message integrity, and message ordering.

Furthermore, endpoints in the group locally perform error handling and processing of invalid messages according to the same principles adopted in [I-D.ietf-core-object-security]. However, a receiver endpoint MUST stop processing and silently reject any message which is malformed and does not follow the format specified in Section 3, without sending back any error message. This prevents listener endpoints from sending multiple error messages to a multicaster endpoint, so avoiding the risk of flooding and possibly congesting the group.

4.1. Protecting the Request

A multicaster endpoint transmits a secure group request as described in Section 8.1 of [I-D.ietf-core-object-security], with the following modifications.

1. The multicaster endpoint stores the association Token - Group Identifier. That is, it SHALL be able to find the correct Security Context used to protect the group request and verify the response(s) by using the CoAP Token used in the message exchange.

2. The multicaster computes the COSE object as defined in Section 3 of this specification.

4.2. Verifying the Request

Upon receiving a secure group request, a listener endpoint proceeds as described in Section 8.2 of [I-D.ietf-core-object-security], with the following modifications.

1. The listener endpoint retrieves the Group Identifier from the 'kid context' parameter of the received COSE object. Then, it uses the Group Identifier together with the destination IP address of the group request to identify the correct group's Security Context.

2. The listener endpoint retrieves the Sender ID from the "kid" parameter of the received COSE object. Then, the Sender ID is used to retrieve the correct Recipient Context associated to the multicaster endpoint and used to process the group request. When receiving a secure group request message from that multicaster endpoint for the first time, the listener endpoint creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], and includes the multicaster endpoint's public key.

3. The listener endpoint retrieves the corresponding public key of the multicaster endpoint from the associated Recipient Context. Then, it verifies the counter signature and decrypts the group request.

4.3. Protecting the Response

A listener endpoint that has received a secure group request may reply with a secure response, which is protected as described in Section 8.3 of [I-D.ietf-core-object-security], with the following modifications.

1. The listener endpoint computes the COSE object as defined in Section 3 of this specification.

4.4. Verifying the Response

Upon receiving a secure response message, a multicaster endpoint proceeds as described in Section 8.4 of [I-D.ietf-core-object-security], with the following modifications.

1. The multicaster endpoint retrieves the Security Context by using the Token of the received response message.

2. The multicaster endpoint retrieves the Sender ID from the "kid" parameter of the received COSE object. Then, the Sender ID is

used to retrieve the correct Recipient Context associated to the listener endpoint and used to process the response message. When receiving a secure response message from that listener endpoint for the first time, the multicaster endpoint creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], and includes the listener endpoint's public key.

3. The multicaster endpoint retrieves the corresponding public key of the listener endpoint from the associated Recipient Context. Then, it verifies the counter signature and decrypts the response message.

The mapping between response messages from listener endpoints and the associated group request from a multicaster endpoint relies on the pair (Sender ID, Partial IV) associated to the secure group request. This is used by listener endpoints as part of the Additional Authenticated Data when protecting their own response message, as described in Section 3.

5. Synchronization of Sequence Numbers

Upon joining the group, new listeners are not aware of the sequence number values currently used by different multicasters to transmit group requests. This means that, when such listeners receive a secure group request from a given multicaster for the first time, they are not able to verify if that request is fresh and has not been replayed. The same holds when a listener endpoint loses synchronization with sequence numbers of multicasters, for instance after a device reboot.

The exact way to address this issue depends on the specific use case and its synchronization requirements. The list of methods to handle synchronization of sequence numbers is part of the group communication policy, and different listener endpoints can use different methods. Appendix E describes three possible approaches that can be considered.

6. Responsibilities of the Group Manager

The Group Manager is responsible for performing the following tasks:

- Creating and managing OSCORE groups. This includes the assignment of a Group ID to every newly created group, as well as ensuring uniqueness of Group IDs within the set of its OSCORE groups.

- Defining policies for authorizing the joining of its OSCORE groups. Such policies can be enforced by a third party, which is

in a trust relation with the Group Manager and enforces join policies on behalf of the Group Manager.

- Driving the join process to add new endpoints as group members.

- Establishing Security Common Contexts and providing them to authorized group members during the join process, together with a corresponding Security Sender Context.

- Generating and managing Endpoint IDs within its OSCORE groups, as well as assigning and providing them to new endpoints during the join process. This includes ensuring uniqueness of Endpoints IDs within each of its OSCORE groups.

- Defining a set of supported signature algorithms as part of the communication policy of each of its OSCORE groups, and signalling it to new endpoints during the join process.

- Defining the methods to handle loss of synchronization with sequence numbers as part of the communication policy of each of its OSCORE groups, and signaling the one(s) to use to new endpoints during the join process.

- Renewing the Security Context of an OSCORE group upon membership change, by revoking and renewing common security parameters and keying material (rekeying).

- Providing the management keying material that a new endpoint requires to participate in the rekeying process, consistently with the key management scheme used in the group joined by the new endpoint.

- Updating the Group ID of its OSCORE groups, upon renewing the respective Security Context.

The Group Manager may additionally be responsible for the following tasks:

- Acting as trusted key repository, in order to store the public keys of the members of its OSCORE groups, and provide such public keys to other members of the same group upon request. This specification recommends that the Group Manager is entrusted to perform this task.

- Acting as network router device where endpoints register to correctly receive group messages sent to the multicast IP address of that group.

- Autonomously and locally enforcing access policies to authorize new endpoints to join its OSCORE groups.

7. Security Considerations

The same security considerations from OSCORE (Section 11 of [I-D.ietf-core-object-security]) apply to this specification. Additional security aspects to be taken into account are discussed below.

7.1. Group-level Security

The approach described in this document relies on commonly shared group keying material to protect communication within a group. This means that messages are encrypted at a group level (group-level data confidentiality), i.e. they can be decrypted by any member of the group, but not by an external adversary or other external entities.

In addition, it is required that all group members are trusted, i.e. they do not forward the content of group messages to unauthorized entities. However, in many use cases, the devices in the group belong to a common authority and are configured by a commissioner (see Appendix B).

8. IANA Considerations

This document has no actions for IANA.

9. Acknowledgments

The authors sincerely thank Stefan Beck, Rolf Blom, Carsten Bormann, Esko Dijk, Klaus Hartke, Richard Kelsey, John Mattsson, Jim Schaad, Ludwig Seitz and Peter van der Stok for their feedback and comments.

The work on this document has been partly supported by the EIT-Digital High Impact Initiative ACTIVE.

10. References

10.1. Normative References

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-08 (work in progress), January 2018.

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

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

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

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

10.2. Informative References

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profiles for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-02 (work in progress), October 2017.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-10 (work in progress), February 2018.

[I-D.ietf-ace-oscore-profile]
Seitz, L., Palombini, F., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-00 (work in progress), December 2017.

[I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-00 (work in progress), October 2017.

[I-D.palombini-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-palombini-ace-key-groupcomm-00 (work in progress), March 2018.

[I-D.somaraju-ace-multicast]
Somaraju, A., Kumar, S., Tschofenig, H., and W. Werner, "Security for Low-Latency Group Communication", draft-somaraju-ace-multicast-02 (work in progress), October 2016.

[I-D.tiloca-ace-oscoap-joining]
Tiloca, M. and J. Park, "Joining of OSCORE multicast groups in ACE", draft-tiloca-ace-oscoap-joining-02 (work in progress), October 2017.

[RFC2093] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification", RFC 2093, DOI 10.17487/RFC2093, July 1997, <https://www.rfc-editor.org/info/rfc2093>.

[RFC2094] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, DOI 10.17487/RFC2094, July 1997, <https://www.rfc-editor.org/info/rfc2094>.

[RFC2627] Wallner, D., Harder, E., and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, DOI 10.17487/RFC2627, June 1999, <https://www.rfc-editor.org/info/rfc2627>.

[RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, DOI 10.17487/RFC3376, October 2002, <https://www.rfc-editor.org/info/rfc3376>.

[RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, DOI 10.17487/RFC3740, March 2004, <https://www.rfc-editor.org/info/rfc3740>.

[RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <https://www.rfc-editor.org/info/rfc3810>.

[RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, DOI 10.17487/RFC4046, April 2005, <https://www.rfc-editor.org/info/rfc4046>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC4535] Harney, H., Meth, U., Colegrove, A., and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol", RFC 4535, DOI 10.17487/RFC4535, June 2006, <https://www.rfc-editor.org/info/rfc4535>.

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

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

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

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

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

Appendix A. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the approach described in this document.

A.1. Assumptions

The following assumptions are assumed to be already addressed and are out of the scope of this document.

- Multicast communication topology: this document considers both 1-to-N (one multicaster and multiple listeners) and M-to-N (multiple multicasters and multiple listeners) communication topologies. The 1-to-N communication topology is the simplest group communication scenario that would serve the needs of a typical low-power and lossy network (LLN). Examples of use cases that benefit from secure group communication are provided in Appendix B.

- Group size: security solutions for group communication should be able to adequately support different and possibly large groups. The group size is the current number of members in a group. In the use cases mentioned in this document, the number of multicasters (normally the controlling devices) is expected to be much smaller than the number of listeners (i.e. the controlled devices). A security solution for group communication that supports 1 to 50 multicasters would be able to properly cover the group sizes required for most use cases that are relevant for this document. The maximum group size is expected to be in the range of 2 to 100 devices. Groups larger than that should be divided into smaller independent groups, e.g. by grouping lights in a building on a per floor basis.

- Communication with the Group Manager: an endpoint must use a secure dedicated channel when communicating with the Group Manager, even when not registered as group member. In particular, communications with the Group Manager occuring during the join process to become a group member must also be secured.

- Establishment and management of Security Contexts: an OSCORE Security Context must be established among the group members. In particular, a Common Context must be provided to a new joining endpoint together with a corresponding Sender Context. On the other hand, Recipient Contexts are locally and individually derived by each group member. A secure mechanism must be used to generate, revoke and (re-)distribute keying material, multicast security policies and security parameters in the group. The actual establishment and management of the Security Context is out

of the scope of this document, and it is anticipated that an activity in IETF dedicated to the design of a generic key management scheme will include this feature, preferably based on [RFC3740][RFC4046][RFC4535].

- Multicast data security ciphersuite: all group members must agree on a ciphersuite to provide authenticity, integrity and confidentiality of messages in the group. The ciphersuite is specified as part of the Security Context.

- Backward security: a new device joining the group should not have access to any old Security Contexts used before its joining. This ensures that a new group member is not able to decrypt confidential data sent before it has joined the group. The adopted key management scheme should ensure that the Security Context is updated to ensure backward confidentiality. The actual mechanism to update the Security Context and renew the group keying material upon a group member's joining has to be defined as part of the group key management scheme.

- Forward security: entities that leave the group should not have access to any future Security Contexts or message exchanged within the group after their leaving. This ensures that a former group member is not able to decrypt confidential data sent within the group anymore. Also, it ensures that a former member is not able to send encrypted and/or integrity protected messages to the group anymore. The actual mechanism to update the Security Context and renew the group keying material upon a group member's leaving has to be defined as part of the group key management scheme.

A.2. Security Objectives

The approach described in this document aims at fulfilling the following security objectives:

- Data replay protection: replayed group request messages or response messages must be detected.

- Group-level data confidentiality: messages sent within the group shall be encrypted if privacy sensitive data is exchanged within the group. This document considers group-level data confidentiality since messages are encrypted at a group level, i.e. in such a way that they can be decrypted by any member of the group, but not by an external adversary or other external entities.

- Source authentication: messages sent within the group shall be authenticated. That is, it is essential to ensure that a message

is originated by a member of the group in the first place, and in particular by a specific member of the group.

- Message integrity: messages sent within the group shall be integrity protected. That is, it is essential to ensure that a message has not been tampered with by an external adversary or other external entities which are not group members.

- Message ordering: it must be possible to determine the ordering of messages coming from a single sender endpoint. In accordance with OSCORE [I-D.ietf-core-object-security], this results in providing relative freshness of group requests and absolute freshness of responses. It is not required to determine ordering of messages from different sender endpoints.

Appendix B. List of Use Cases

Group Communication for CoAP [RFC7390] provides the necessary background for multicast-based CoAP communication, with particular reference to low-power and lossy networks (LLNs) and resource constrained environments. The interested reader is encouraged to first read [RFC7390] to understand the non-security related details. This section discusses a number of use cases that benefit from secure group communication. Specific security requirements for these use cases are discussed in Appendix A.

- Lighting control: consider a building equipped with IP-connected lighting devices, switches, and border routers. The devices are organized into groups according to their physical location in the building. For instance, lighting devices and switches in a room or corridor can be configured as members of a single group. Switches are then used to control the lighting devices by sending on/off/dimming commands to all lighting devices in a group, while border routers connected to an IP network backbone (which is also multicast-enabled) can be used to interconnect routers in the building. Consequently, this would also enable logical groups to be formed even if devices in the lighting group may be physically in different subnets (e.g. on wired and wireless networks). Connectivity between lighting devices may be realized, for instance, by means of IPv6 and (border) routers supporting 6LoWPAN [RFC4944][RFC6282]. Group communication enables synchronous operation of a group of connected lights, ensuring that the light preset (e.g. dimming level or color) of a large group of luminaires are changed at the same perceived time. This is especially useful for providing a visual synchronicity of light effects to the user. As a practical guideline, events within a 200 ms interval are perceived as simultaneous by humans, which is necessary to ensure in many setups. Devices may reply back to the

switches that issue on/off/dimming commands, in order to report about the execution of the requested operation (e.g. OK, failure, error) and their current operational status. In a typical lighting control scenario, a single switch is the only entity responsible for sending commands to a group of lighting devices. In more advanced lighting control use cases, a M-to-N communication topology would be required, for instance in case multiple sensors (presence or day-light) are responsible to trigger events to a group of lighting devices. Especially in professional lighting scenarios, the roles of multicaster and listener are configured by the lighting commissioner, and devices strictly follow those roles.

- Integrated building control: enabling Building Automation and Control Systems (BACSs) to control multiple heating, ventilation and air-conditioning units to pre-defined presets. Controlled units can be organized into groups in order to reflect their physical position in the building, e.g. devices in the same room can be configured as members of a single group. As a practical guideline, events within intervals of seconds are typically acceptable. Controlled units are expected to possibly reply back to the BACS issuing control commands, in order to report about the execution of the requested operation (e.g. OK, failure, error) and their current operational status.

- Software and firmware updates: software and firmware updates often comprise quite a large amount of data. This can overload a LLN that is otherwise typically used to deal with only small amounts of data, on an infrequent base. Rather than sending software and firmware updates as unicast messages to each individual device, multicasting such updated data to a larger group of devices at once displays a number of benefits. For instance, it can significantly reduce the network load and decrease the overall time latency for propagating this data to all devices. Even if the complete whole update process itself is secured, securing the individual messages is important, in case updates consist of relatively large amounts of data. In fact, checking individual received data piecemeal for tampering avoids that devices store large amounts of partially corrupted data and that they detect tampering hereof only after all data has been received. Devices receiving software and firmware updates are expected to possibly reply back, in order to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.

- Parameter and configuration update: by means of multicast communication, it is possible to update the settings of a group of similar devices, both simultaneously and efficiently. Possible

parameters are related, for instance, to network load management or network access controls. Devices receiving parameter and configuration updates are expected to possibly reply back, to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.

- Commissioning of LLNs systems: a commissioning device is responsible for querying all devices in the local network or a selected subset of them, in order to discover their presence, and be aware of their capabilities, default configuration, and operating conditions. Queried devices displaying similarities in their capabilities and features, or sharing a common physical location can be configured as members of a single group. Queried devices are expected to reply back to the commissioning device, in order to notify their presence, and provide the requested information and their current operational status.

- Emergency multicast: a particular emergency related information (e.g. natural disaster) is generated and multicast by an emergency notifier, and relayed to multiple devices. The latters may reply back to the emergency notifier, in order to provide their feedback and local information related to the ongoing emergency. This kind of setups should additionally rely on a fault tolerance multicast algorithm, such as MPL.

Appendix C. Example of Group Identifier Format

This section provides an example of how the Group Identifier (Gid) can be specifically formatted. That is, the Gid can be composed of two parts, namely a Group Prefix and a Group Epoch.

The Group Prefix is uniquely defined in the set of all the groups associated to the same Group Manager. The choice of the Group Prefix for a given group's Security Context is application specific. A Group Prefix is random, constant over time, and long enough to achieve a negligible probability of collisions between Group Identifiers from different Group Managers. The size of the Group Prefix directly impact on the maximum number of distinct groups under the same Group Manager.

The Group Epoch is set to 0 upon the group's initialization, and is incremented by 1 upon completing each renewal of the Security Context and keying material in the group (see Section 2.1). In particular, once a new Master Secret has been distributed to the group, all the group members increment by 1 the Group Epoch in the Group Identifier of that group.

As an example, a 3-byte Group Identifier can be composed of: i) a 1-byte Group Prefix '0xb1' interpreted as a raw byte string; and ii) a 2-byte Group Epoch interpreted as an unsigned integer ranging from 0 to 65535. Then, after having established the Security Common Context 61532 times in the group, its Group Identifier will assume value '0xb1f05c'.

Appendix D. Set-up of New Endpoints

An endpoint joins a group by explicitly interacting with the responsible Group Manager. Communications between a joining endpoint and the Group Manager rely on the CoAP protocol and must be secured. Specific details on how to secure communications between joining endpoints and a Group Manager are out of scope.

In order to receive multicast messages sent to the group, a joining endpoint has to register with a network router device [RFC3376][RFC3810], signaling its intent to receive packets sent to the multicast IP address of that group. As a particular case, the Group Manager can also act as such a network router device. Upon joining the group, endpoints are not required to know how many and what endpoints are active in the same group.

Furthermore, in order to participate in the secure group communication, an endpoint needs to be properly initialized upon joining the group. In particular, the Group Manager provides keying material and parameters to a joining endpoint, which can then initialize its own Security Context (see Section 2).

The following Appendix D.1 provides an example describing how such information can be provided to an endpoint upon joining a group through the responsible Group Manager. Then, Appendix D.2 discusses how public keys of group members can be handled and made available to group members. Finally, Appendix D.3 overviews how the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz] can be possibly used to support such a join process.

D.1. Join Process

An endpoint requests to join a group by sending a confirmable CoAP POST request to the Group Manager responsible for that group. This join request can reflect the format of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm]. Besides, it can be addressed to a CoAP resource associated to that group and carries the following information.

- Group identifier: the Group Identifier (Gid) of the group, as known to the joining endpoint at this point in time. This may not fully coincide with the Gid currently associated to the group, e.g. if it includes a dynamic component. This information can be mapped to the first element of the "scope" parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].

- Role: the exact role of the joining endpoint in the group. Possible values are: "multicaster", "listener", "pure listener", "multicaster and listener", or "multicaster and pure listener". This information can be mapped to the second element of the "scope" parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].

- Retrieval flag: indication of interest to receive the public keys of the endpoints currently in the group, as included in the following join response. This flag must not be present if the Group Manager is not configured to store the public keys of group members, or if the joining endpoint is configured exclusively as pure listener for the group to join. This information can be mapped to the "get_pub_keys" parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].

- Identity credentials: information elements to enforce source authentication of group messages from the joining endpoint, such as its public key. The exact content depends on whether the Group Manager is configured to store the public keys of group members. If this is the case, this information is omitted if it has been provided to the same Group Manager upon previously joining the same or a different group under its control. This information is also omitted if the joining endpoint is configured exclusively as pure listener for the joined group. Appendix D.2 discusses additional details on provisioning of public keys and other information to enforce source authentication of joining endpoints's messages. This information can be mapped to the "client_cred" parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].

The Group Manager must be able to verify that the joining endpoint is authorized to become a member of the group. To this end, the Group Manager can directly authorize the joining endpoint, or expect it to provide authorization evidence previously obtained from a trusted entity. Appendix D.3 describes how this can be achieved by leveraging the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

In case of successful authorization check, the Group Manager generates an Endpoint ID assigned to the joining endpoint, before proceeding with the rest of the join process. Instead, in case the authorization check fails, the Group Manager aborts the join process. Further details about the authorization of joining endpoint are out of scope.

As discussed in Section 2.1, it is recommended that the Security Context is renewed before the joining endpoint receives the group keying material and becomes a new active member of the group. This is achieved by securely distributing a new Master Secret and a new Group Identifier to the endpoints currently present in the same group.

Once renewed the Security Context in the group, the Group Manager replies to the joining endpoint with a CoAP response carrying the following information. This join response can reflect the format of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].

- Security Common Context: the OSCORE Security Common Context associated to the joined group (see Section 2). This information can be mapped to the "key" parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].

- Endpoint ID: the Endpoint ID associated to the joining endpoint. This information is not included in case "Role" in the join request is equal to "pure listener". This information can be mapped to the "clientID" parameter within the "key" parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].

- Member public keys: the public keys of the endpoints currently present in the group. This includes: the public keys of the non-pure listeners currently in the group, if the joining endpoint is configured (also) as multicaster; and the public keys of the multicasters currently in the group, if the joining endpoint is configured (also) as listener or pure listener. This information is omitted in case the Group Manager is not configured to store the public keys of group members or if the "Retrieval flag" was not present in the join request. Appendix D.2 discusses additional details on provisioning public keys upon joining the group and on retrieving public keys of group members. This information can be mapped to the "pub_keys" parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].

- Group policies: a list of key words indicating the particular policies enforced in the group. This includes, for instance, the list of supported signature algorithms and the method to achieve synchronization of sequence numbers among group members (see Appendix E). This information can be mapped to the "group_policies" parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].

- Management keying material: the set of administrative keying material used to participate in the group rekeying process run by the Group Manager (see Section 2.1). The specific elements of this management keying material depend on the group rekeying protocol used in the group. For instance, this can simply consist in a group key encryption key and a pairwise symmetric key shared between the joining endpoint and the Group Manager, in case GKMP [RFC2093][RFC2094] is used. Instead, if key-tree based rekeying protocols like LKH [RFC2627] are used, it can consist in the set of symmetric keys associated to the key-tree leaf representing the group member up to the key-tree root representing the group key encryption key. This information can be mapped to the "mgt_key_material" parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].

D.2. Provisioning and Retrieval of Public Keys

As mentioned in Section 6, it is recommended that the Group Manager acts as trusted key repository, so storing public keys of group members and providing them to other members of the same group upon request. In such a case, a joining endpoint provides its own public key to the Group Manager, as "Identity credentials" of the join request, when joining the group (see Appendix D.1).

After that, the Group Manager should verify that the joining endpoint actually owns the associated private key, for instance by performing a proof-of-possession challenge-response, whose details are out of scope. In case of failure, the Group Manager performs up to a pre-defined maximum number of retries, after which it aborts the join process.

In case of successful challenge-response, the Group Manager stores the received public key as associated to the joining endpoint and its Endpoint ID. From then on, that public key will be available for secure and trusted delivery to other endpoints in the group. Finally, the Group Manager sends the join response to the joining endpoint, as described in Appendix D.1.

The joining endpoint does not have to provide its own public key if that already occurred upon previously joining the same or a different group under the same Group Manager. However, separately for each group under its control, the Group Manager maintains an updated list of active Endpoint IDs associated to the respective endpoint's public key.

Instead, in case the Group Manager does not act as trusted key repository, the following exchange with the Group Manager can occur during the join process.

1. The joining endpoint signs its own certificate by using its own private key. The certificate includes also the identifier of the issuer Certification Authority (CA). There is no restriction on the Certificate Subject included in the joining endpoint's certificate.

2. The joining endpoint specifies the signed certificate as "Identity credentials" in the join request (Appendix D.1). The joining endpoint can optionally specify also a list of public key repositories storing its own certificate. In such a case, this information can be mapped to the "pub_keys_repos" parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].

3. When processing the join request, the Group Manager first validates the certificate by verifying the signature of the issuer CA, and then verifies the signature of the joining endpoint.

4. The Group Manager stores the association between the Certificate Subject of the joining endpoint's certificate and the pair {Group ID, Endpoint ID of the joining endpoint}. If received from the joining endpoint, the Group Manager also stores the list of public key repositories storing the certificate of the joining endpoint.

When a group member X wants to retrieve the public key of another group member Y in the same group, the endpoint X proceeds as follows.

1. The endpoint X contacts the Group Manager, specifying the pair {Group ID, Endpoint ID of the endpoint Y}.

2. The Group Manager provides the endpoint X with the Certificate Subject CS from the certificate of endpoint Y. If available, the Group Manager provides the endpoint X also with the list of public key repositories storing the certificate of the endpoint Y.

3. The endpoint X retrieves the certificate of the endpoint X from a key repository storing it, by using the Certificate Subject CS.

D.3. Group Joining Based on the ACE Framework

The join process to register an endpoint as a new member of a group can be based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz], built on re-use of OAuth 2.0 [RFC6749].

In particular, the approach described in [I-D.tiloca-ace-oscoap-joining] uses the ACE framework to delegate the authentication and authorization of joining endpoints to an Authorization Server in a trust relation with the Group Manager. At the same time, it allows a joining endpoint to establish a secure channel with the Group Manager, by leveraging protocol-specific profiles of ACE, such as [I-D.ietf-ace-oscore-profile] and [I-D.ietf-ace-dtls-authorize], to achieve communication security, proof-of-possession and server authentication.

More specifically and with reference to the terminology defined in OAuth 2.0:

- The joining endpoint acts as Client;
- The Group Manager acts as Resource Server, with different CoAP resources for different groups it is responsible for;
- An Authorization Server enables and enforces authorized access of the joining endpoint to the Group Manager and its CoAP resources paired with groups to join.

Messages exchanged among the participants follow the formats defined in [I-D.palombini-ace-key-groupcomm]. Both the joining endpoint and the Group Manager have to adopt secure communication also for any message exchange with the Authorization Server. To this end, different alternatives are possible, such as OSCORE, DTLS [RFC6347] or IPsec [RFC4301].

Appendix E. Examples of Synchronization Approaches

This section describes three possible approaches that can be considered by listener endpoints to synchronize with sequence numbers of multicasters.

E.1. Best-Effort Synchronization

Upon receiving a multicast request from a multicaster, a listener endpoint does not take any action to synchonize with the sequence number of that multicaster. This provides no assurance at all as to message freshness, which can be acceptable in non-critical use cases.

E.2. Baseline Synchronization

Upon receiving a multicast request from a given multicaster for the first time, a listener endpoint initializes its last-seen sequence number in its Recipient Context associated to that multicaster. However, the listener drops the multicast request without delivering it to the application layer. This provides a reference point to identify if future group requests from the same multicaster are fresher than the last one received.

A replay time interval exists, between when a possibly replayed message is originally transmitted by a given multicaster and the first authentic fresh message from that same multicaster is received. This can be acceptable for use cases where listener endpoints admit such a trade-off between performance and assurance of message freshness.

E.3. Challenge-Response Synchronization

A listener endpoint performs a challenge-response exchange with a multicaster, by using the Repeat Option for CoAP described in Section 2 of [I-D.ietf-core-echo-request-tag].

That is, upon receiving a group request from a particular multicaster for the first time, the listener processes the message as described in Section 4.2 of this specification, but, even if valid, does not deliver it to the application. Instead, the listener replies to the multicaster with a 4.03 Forbidden response message including a Repeat Option, and stores the option value included therein.

Upon receiving a 4.03 Forbidden response that includes a Repeat Option and originates from a verified group member, a multicaster sends a request as a unicast message addressed to the same listener, echoing the Repeat Option value. In particular, the multicaster does not necessarily resend the same group request, but can instead send a more recent one, if the application permits it. This makes it possible for the multicaster to not retain previously sent group requests for full retransmission, unless the application explicitly requires otherwise. In either case, the multicaster uses the sequence number value currently stored in its own Sender Context. If the multicaster stores group requests for possible retransmission

with the Repeat Option, it should not store a given request for longer than a pre-configured time interval. Note that the unicast request echoing the Repeat Option is correctly treated and processed as a group message, since the 'kid context' field including the Group Identifier of the OSCORE group is still present in the Object-Security Option as part of the COSE object (see Section 3).

Upon receiving the unicast request including the Repeat Option, the listener verifies that the option value equals the stored and previously sent value; otherwise, the request is silently discarded. Then, the listener verifies that the unicast request has been received within a pre-configured time interval, as described in [I-D.ietf-core-echo-request-tag]. In such a case, the request is further processed and verified; otherwise, it is silently discarded. Finally, the listener updates the Recipient Context associated to that multicaster, by setting the Replay Window according to the Sequence Number from the unicast request conveying the Repeat Option. The listener either delivers the request to the application if it is an actual retransmission of the original one, or discards it otherwise. Mechanisms to signal whether the resent request is a full retransmission of the original one are out of the scope of this specification.

In case it does not receive a valid unicast request including the Repeat Option within the configured time interval, the listener endpoint should perform the same challenge-response upon receiving the next multicast request from that same multicaster.

A listener should not deliver group requests from a given multicaster to the application until one valid request from that same multicaster has been verified as fresh, as conveying an echoed Repeat Option [I-D.ietf-core-echo-request-tag]. Also, a listener may perform the challenge-response described above at any time, if synchronization with sequence numbers of multicasters is (believed to be) lost, for instance after a device reboot. It is the role of the application to define under what circumstances sequence numbers lose synchronization. This can include a minimum gap between the sequence number of the latest accepted group request from a multicaster and the sequence number of a group request just received from the same multicaster. A multicaster has to be always ready to perform the challenge-response based on the Repeat Option in case a listener starts it.

Note that endpoints configured as pure listeners are not able to perform the challenge-response described above, as they do not store a Sender Context to secure the 4.03 Forbidden response to the multicaster. Therefore, pure listeners should adopt alternative

approaches to achieve and maintain synchronization with sequence numbers of multicasters.

This approach provides an assurance of absolute message freshness. However, it can result in an impact on performance which is undesirable or unbearable, especially in large groups where many endpoints at the same time might join as new members or lose synchronization.

Appendix F. No Verification of Signatures

There are some application scenarios using group communication that have particularly strict requirements. One example of this is the requirement of low message latency in non-emergency lighting applications [I-D.somaraju-ace-multicast]. For those applications which have tight performance constraints and relaxed security requirements, it can be inconvenient for some endpoints to verify digital signatures in order to assert source authenticity of received group messages. In other cases, the signature verification can be deferred or only checked for specific actions. For instance, a command to turn a bulb on where the bulb is already on does not need the signature to be checked. In such situations, the counter signature needs to be included anyway as part of the group message, so that an endpoint that needs to validate the signature for any reason has the ability to do so.

In this specification, it is NOT RECOMMENDED that endpoints do not verify the counter signature of received group messages. However, it is recognized that there may be situations where it is not always required. The consequence of not doing the signature validation is that security in the group is based only on the group-authenticity of the shared keying material used for encryption. That is, endpoints in the group have evidence that a received message has been originated by a group member, although not specifically identifiable in a secure way. This can violate a number of security requirements, as the compromise of any element in the group means that the attacker has the ability to control the entire group. Even worse, the group may not be limited in scope, and hence the same keying material might be used not only for light bulbs but for locks as well. Therefore, extreme care must be taken in situations where the security requirements are relaxed, so that deployment of the system will always be done safely.

Authors' Addresses

Marco Tiloca
RISE SICS AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Goeran Selander
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

CoRE
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

K. Lynn
P. van der Stok
Consultants
M. Koster
SmartThings
C. Amsuess
Energy Harvesting Solutions

March 05, 2018

CoRE Resource Directory: DNS-SD mapping
draft-ietf-core-rd-dns-sd-01

Abstract

Resource and service discovery are complimentary. Resource discovery provides fine-grained detail about the content of a server, while service discovery can provide a scalable method to locate servers in large networks. This document defines a method for mapping between CoRE Link Format attributes and DNS-Based Service Discovery fields to facilitate the use of either method to locate RESTful service interfaces (APIs) in mixed HTTP/CoAP environments.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 6, 2018.

Table of Contents

1. Introduction 2
1.1. Terminology 3
1.2. Resource Discovery 3
1.3. Resource Directories 4
1.4. DNS-Based Service Discovery 4
2. New Link-Format Attributes 5
2.1. Resource Instance attribute "ins" 6
2.2. Export attribute "exp" 6
3. Mapping CoRE Link Attributes to DNS-SD Record Fields 6
3.1. Mapping Resource Instance attribute "ins" to <Instance> . 6
3.2. Mapping Resource Type attribute "rt" to <ServiceType> . . 7
3.3. Domain mapping 7
3.4. TXT Record key=value strings 7
3.5. Importing resource links into DNS-SD 8
4. Examples 9
4.1. DNS entries 9
5. IANA considerations 9
6. Security considerations 9
7. References 9
7.1. Normative References 9
7.2. Informative References 10
Acknowledgements 11
Authors' Addresses 11

1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained devices (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 main deliverable of CoRE is the Constrained Application Protocol (CoAP) specification [RFC7252].

Automated discovery of resources hosted by a constrained server is critical in M2M applications where human intervention is minimal and static interfaces result in brittleness. 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 end-point's IP address, port number, and protocol. This definition may be extended to include multi-function devices, where the result of the discovery process may include a path to a resource representing a RESTful service interface and possibly a reference to a description of the interface such as a JSON Hyper-Schema document [I-D.handrews-json-schema-hyperschema].

Resource and service discovery are complimentary 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 (DNS-SD) [RFC6763] fields that permits discovery of CoAP services by either means. It also addresses the CoRE charter goal to interoperate with DNS-SD.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [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 {-link} 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 also makes use of the terminology of [I-D.ietf-core-resource-directory].

1.2. Resource Discovery

The main function of Resource Discovery is to provide Universal Resource Identifiers (URIs, also called "links") for the resources hosted by the server, complemented by attributes about those resources and perhaps additional link relations. In CoRE this collection of links and attributes is itself a resource (as opposed to HTTP headers delivered with a specific resource).

[RFC6690] specifies a link format for use in CoRE Resource Discovery by extending the HTTP Link Header Format [RFC8288] to describe these link descriptions. The CoRE Link Format is carried as a payload and is assigned an Internet media type. A well-known URI "/.well-known/core" is defined as a default entry-point for requesting the list of links about resources hosted by a server, and thus performing CoRE Resource Discovery.

Resource Discovery can be performed either via unicast or multicast. When a server's IP address is already known, either a priori or resolved via the Domain Name System (DNS) [RFC1034][RFC1035], unicast discovery is performed in order to locate a URI for the resource of interest. This is performed using a GET to /.well-known/core on the server, which returns a payload in the CoRE Link Format. A client would then match the appropriate Resource Type, Interface Description, and possible Content-Type [RFC2045] for its application. These attributes may also be included in the query string in order to filter the number of links returned in a response.

1.3. Resource Directories

In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, limited bandwidth, or networks where multicast traffic is inefficient. These problems can be solved by deploying a network element called a Resource Directory (RD), which hosts descriptions of resources held on other servers (referred to as "end-points") and allows lookups to be performed for those resources. An end-point is a web server associated with specific IP address and port; thus a physical device may host one or more end-points. End-points may also act as clients.

The Resource Directory implements a set of REST interfaces for end-points to register and maintain sets of Web Links, called resource directory entries. [I-D.ietf-core-resource-directory] specifies the web interfaces that an RD supports in order for web servers to discover the RD and to register, maintain, lookup and remove resource descriptions; for the RD to validate entries; and for clients to lookup resources from the RD. Furthermore, new link attributes useful in conjunction with an RD are defined.

1.4. 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 bytes and are of the form:

Service Name = <Instance>.<ServiceType>.<Domain>

The service name is the label of SRV/TXT resource records. The SRV RR specifies the host and the port of the endpoint. The TXT RR provides additional information in the form of key/value pairs.

The <Domain> part of the service name is identical to the global (DNS subdomain) part of the authority in URIs that identify the resources on an individual server or group 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 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". Only the rightmost pair of labels is used in SRV and TXT record names.

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 bytes in length, which is indicated in the resource record header in the DNS response message [RFC6763]. 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). An example string is:

| 0x08 | t | x | t | v | e | r | = | 1 |

2. 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" "=" (ptoken | quoted-string) )
                    ; The token or string is max 63 bytes
link-extension    = ( "exp" )
```

2.1. 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 1.4, 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. This attribute MAY be used as a query parameter in the RD Lookup Function Set defined in Section 7 of [I-D.ietf-core-resource-directory].

2.2. 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. This attribute MAY be used as a query parameter in the RD Lookup Function Set defined in Section 7 of [I-D.ietf-core-resource-directory].

3. Mapping CoRE Link Attributes to DNS-SD Record Fields

3.1. Mapping Resource Instance attribute "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 bytes in length and the entire service name may not exceed 255 bytes.

3.2. Mapping Resource Type attribute "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 bytes 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 bytes. This string is mapped to the DNS-SD <ServiceType> by appending a period followed by the "_sub" label and then appending a period followed by the service type label pair derived as in the previous paragraph. For example, rt="dali.light" is mapped into "light._sub._dali._udp".

The resulting string is used to form labels for DNS-SD records which are stored directly in the DNS.

3.3. Domain mapping

TBD: A method must be specified to determine in which DNS zone the CoAP service should be registered. See, for example, Section 11 in [RFC6763].

3.4. 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].

3.5. 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, using the example lookup location /rd-lookup, shows an agent discovering a resource to be exported:

```
Req: GET /rd-lookup/res?exp

Res: 2.05 Content
<coap://[FDFD::1234]:5683/light/1>;
  exp;rt="dali.light";ins="Spot";
            d="office";ep="node1"
```

The agent subsequently registers the following DNS-SD RRs, assuming a zone name "example.com" prefixed with "office":

```
node1.office.example.com.          IN AAAA        FDFD::1234
_dali._udp.office.example.com      IN PTR
                           Spot._dali._udp.office.example.com
light._sub._dali._udp.example.com  IN PTR
                           Spot._dali._udp.office.example.com
Spot._dali._udp.office.example.com IN SRV  0 0 5683
                           node1.office.example.com.
Spot._dali._udp.office.example.com IN TXT
                           txtver=1;path=/light/1
```

In the above figure the Service Name is chosen as Spot._dali._udp.office.example.com without the light._sub service prefix. An alternative Service Name would be: Spot.light._sub._dali._udp.office.example.com.

4. Examples

4.1. DNS entries

It may be profitable to discover the light groups for applications, which are unaware ot the existence of the RD. An agent needs to query the RD to return all groups which are exported to be inserted into DNS.

```
   Req: GET /rd-lookup/gp?exp

   Res: 2.05 Content
   <coap://[FF05::1]/>;exp;gp="grp_R2-4-015;ins="grp1234";
ep="lm_R2-4-015_wndw";
ep="lm_R2-4-015_door
```

The group with FQDN grp_R2-4-015.bc.example.com can be entered into the DNS by the agent. The accompanying instance name is grp1234. The <ServiceType> is chosen to be _group._udp. The agent enters the following RRs into the DNS.

```
grp_R2-4-015.bc.example.com.        IN AAAA            FF05::1
_group._udp.bc.example.com          IN PTR
                              grp1234._group._udp.bc.example.com
grp1234._group._udp.bc.example.com  IN SRV  0 0 5683
                               grp_R2-4-015_door.bc.example.com.
grp1234._group._udp.bc.example.com  IN TXT
                                     txtver=1;path=/light/grp1
```

From then on, applications unaware of the existence of the RD can use DNS to access the lighting group.

5. IANA considerations

TBD

6. Security considerations

TBD

7. References

7.1. Normative References

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

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

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

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

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

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

7.2. Informative References

[I-D.handrews-json-schema-hyperschema] Andrews, H. and A. Wright, "JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON", draft-handrews-json-schema-hyperschema-01 (work in progress), January 2018.

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-13 (work in progress), March 2018.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/info/rfc5198>.

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

Acknowledgements

This document was split out from [I-D.ietf-core-resource-directory]. Zach Shelby was a co-author of the original version of this draft.

Authors' Addresses

Kerry Lynn
Consultant

Phone: +1 978-460-4253
Email: kerlyn@ieee.org

Peter van der Stok
Consultant

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

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Phone: +1 707-502-5136
Email: Michael.Koster@smartthings.com

Christian Amsuess
Energy Harvesting Solutions
Hollandstr. 12/4
1020
Austria

Phone: +43 664-9790639
Email: c.amsuess@energyharvesting.at

CoRE
Internet-Draft
Intended status: Standards Track
Expires: September 2, 2018

Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
C. Amsuess, Ed.
March 01, 2018

CoRE Resource Directory
draft-ietf-core-resource-directory-13

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 resource 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on September 2, 2018.

Table of Contents

1. Introduction 3
2. Terminology 4
3. Architecture and Use Cases 6
3.1. Principles 6
3.2. Architecture 6
3.3. RD Content Model 8
3.4. Use Case: Cellular M2M 12
3.5. Use Case: Home and Building Automation 13
3.6. Use Case: Link Catalogues 13
4. Finding a Resource Directory 14
4.1. Resource Directory Address Option (RDAO) 15
5. Resource Directory 17
5.1. Payload Content Formats 18
5.2. URI Discovery 18
5.3. Registration 21
5.3.1. Simple Registration 25
5.3.2. Third-party registration 26
5.4. Operations on the Registration Resource 26
5.4.1. Registration Update 27
5.4.2. Registration Removal 30
5.4.3. Read Endpoint Links 31
5.4.4. Update Endpoint Links 32
6. RD Groups 32
6.1. Register a Group 32
6.2. Group Removal 34
7. RD Lookup 35
7.1. Resource lookup 36
7.2. Endpoint and group lookup 36
7.3. Lookup filtering 37
7.4. Lookup examples 39
8. Security Considerations 43

8.1. Endpoint Identification and Authentication 43
8.2. Access Control 44
8.3. Denial of Service Attacks 44
9. IANA Considerations 45
9.1. Resource Types 45
9.2. IPv6 ND Resource Directory Address Option 45
9.3. RD Parameter Registry 45
9.3.1. Full description of the "Endpoint Type" Registration Parameter 47
9.4. "Endpoint Type" (et=) RD Parameter values 47
9.5. Multicast Address Registration 48
10. Examples 48
10.1. Lighting Installation 48
10.1.1. Installation Characteristics 48
10.1.2. RD entries 49
10.2. OMA Lightweight M2M (LWM2M) Example 52
10.2.1. The LWM2M Object Model 53
10.2.2. LWM2M Register Endpoint 54
10.2.3. LWM2M Update Endpoint Registration 56
10.2.4. LWM2M De-Register Endpoint 56
11. Acknowledgments 56
12. Changelog 56
13. References 62
13.1. Normative References 62
13.2. Informative References 63
Appendix A. Web links and the Resource Directory 64
A.1. A simple example 64
A.1.1. Resolving the URIs 64
A.1.2. Interpreting attributes and relations 65
A.2. A slightly more complex example 65
A.3. Enter the Resource Directory 66
A.4. A note on differences between link-format and Link headers 67
Appendix B. Syntax examples for Protocol Negotiation 68
Authors' Addresses 69

1. Introduction

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

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically

called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. However, [RFC6690] only describes how to discover resources from the web server that hosts them by querying "/.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 [RFC3986], [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:

resolve against
: The expression "a URI-reference is _resolved against_ a base URI" is used to describe the process of [RFC3986] Section 5.2. Noteworthy corner cases are that resolving an absolute URI against any base URI gives the original URI, and that resolving an empty URI reference gives the base URI.

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.

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 have unique names.

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 has a unique name within the associated domain of the registration.

Context
: A Context is a base URL that gives scheme and (typically) authority information about an Endpoint. The Context of an Endpoint is provided at registration time, and is used by the Resource Directory to resolve relative references inside the registration into absolute URIs.

Directory Resource
: A resource in the Resource Directory (RD) containing registration resources.

Group Resource
: A resource in the RD containing registration resources of the Endpoints that form a group.

Registration Resource
: A resource in the RD that contains information about an Endpoint and its links.

Commissioning Tool
: Commissioning Tool (CT) is a device that assists during the installation of the network by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

RDAO
: Resource Directory Address Option.

3. Architecture and Use Cases

3.1. Principles

The Resource Directory is primarily a tool to make discovery operations more efficient than querying /.well-known/core on all connected device, or across boundaries that would be limiting those operations.

It provides a cache (in the high-level sense, not as defined in [RFC7252]/[RFC2616]) of data that could otherwise only be obtained by directly querying the /.well-known/core resource on the target device, or by accessing those resources with a multicast request.

From that, it follows that only information should be stored in the resource directory that is discovered from querying the described device's /.well-known/core resource directly.

It also follows that data in the resource directory can only be provided by the device whose descriptions are cached or a dedicated Commissioning Tool (CT). These CTs are thought to act on behalf agents too constrained, or generally unable, to present that information themselves. No other client can modify data in the resource directory. Changes in the Resource Directory do not propagate automatically back to its source.

3.2. Architecture

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. 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 registration 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 Groups. The group an endpoint is part of, can be defined by the RD or configured by a Commissioning Tool. This information hierarchy is shown in Figure 2.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Endpoints proactively register and maintain resource directory registration entries on the RD, which are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a resource directory registration entry. It is also possible for an RD to fetch Web Links from endpoints and add them as resource directory registration entries.

At the first registration of a set of entries, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registration entries.

A lookup interface for discovering any of the Web Links held in the RD is provided using the CoRE Link Format.

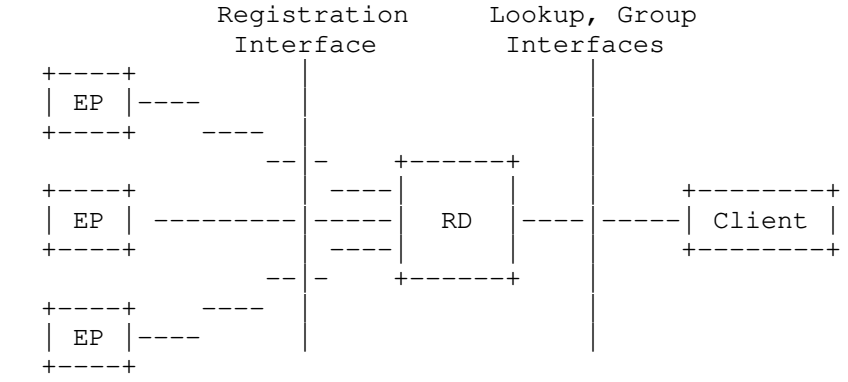


Figure 1: The resource directory architecture.

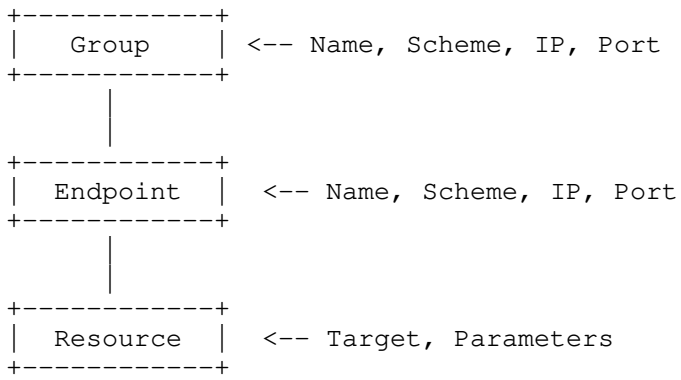


Figure 2: The resource directory information hierarchy.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 3 and Figure 4 model the contents of /.well-known/core and the resource directory respectively, with entity-relationship diagrams [ER]. Entities (rectangles) are used for concepts that exist independently. Attributes (ovals) are used for concepts that exist only in connection with a related entity. Relations (diamonds) give a semantic meaning to the relation between entities. Numbers specify the cardinality of the relations.

Some of the attribute values are URIs. Those values are always full URIs and never relative references in the information model. They can, however, be expressed as relative references in serializations, and often are.

These models provide an abstract view of the information expressed in link-format documents and a Resource Directory. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

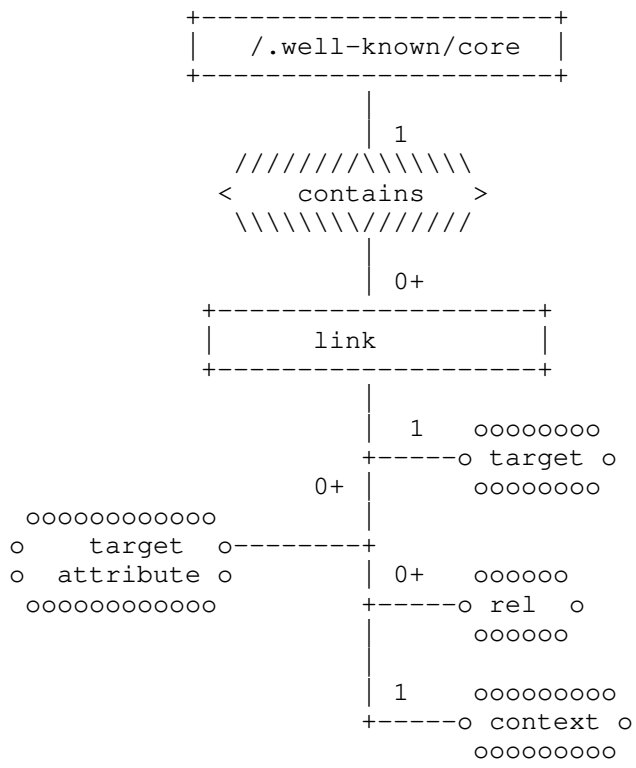


Figure 3: E-R Model of the content of /.well-known/core

The model shown in Figure 3 models the contents of /.well-known/core which contains:

o a set of links belonging to the host

The host is free to choose links it deems appropriate to be exposed in its ".well-known/core". Typically, the links describe resources that are served by the host, but the set can also contain links to resources on other servers (see examples in [RFC6690] page 14). The set does not necessarily contain links to all resources served by the host.

A link has the following attributes (see [RFC5988]):

o Zero or more link relations: They describe relations between the link context and the link target.

In link-format serialization, they are expressed as space-separated values in the "rel" attribute, and default to "hosts".

- A link context URI: It defines the source of the relation, eg. _who_ "hosts" something.

  In link-format serialization, it is expressed in the "anchor" attribute. There, it can be a relative reference, in which case it gets resolved against the URI of the ".well-known/core" document it was obtained from. It defaults to that document's URI.

- A link target URI: It defines the destination of the relation (eg. _what_ is hosted), and is the topic of all target attributes.

  In link-format serialization, it is expressed between angular brackets, and sometimes called the "href".

  If there is an anchor attribute present and the link is serialized in [RFC6690] link format, this document will require that the link is an absolute reference to avoid the ambiguities outlined in Appendix A.4.

  Otherwise, it can be serialized as a relative URI, and gets resolved against the document's URI.

- Other target attributes (eg. resource type (rt), interface (if), cor content-type (ct)). These provide additional information about the target URI.

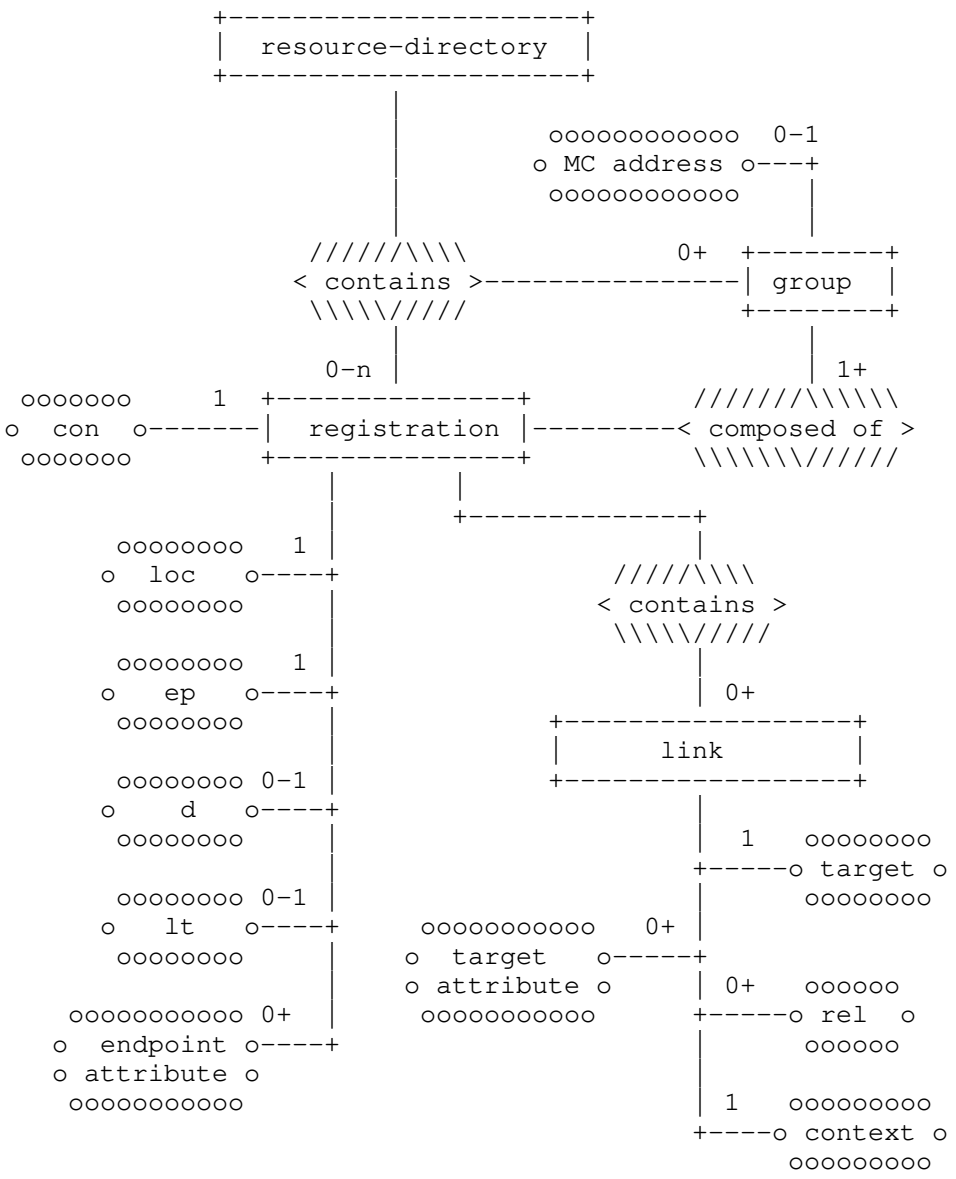


Figure 4: E-R Model of the content of the Resource Directory

The model shown in Figure 4 models the contents of the resource directory which contains in addition to /.well-known/core:

o 0 to n Registration (entries),

- 0 or more Groups

A Group has no or one Multicast address attribute and is composed of 0 or more endpoints. A registration is associated with one endpoint (ep). An endpoint can be part of 0 or more Groups . A registration defines a set of links as defined for /.well-known/core. A Registration has six attributes:

- one ep (endpoint with a unique name)
- one con (a string describing the scheme://authority part)
- one lt (lifetime),
- one loc (location in the RD)
- optional one d (domain for query filtering),
- optional additional endpoint attributes (from Section 9.3)

The cardinality of con is currently 1; future documents are invited to extend the RD specification to support multiple values (eg. [I-D.silverajan-core-coap-protocol-negotiation]). Its value is used as a Base URI when resolving URIs in the links contained in the endpoint.

Links are modelled as they are in Figure 3.

3.4. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded wireless interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. 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.

Imagine a scenario where endpoints installed on vehicles enable tracking of the position of these vehicles for fleet management purposes and allow monitoring of environment parameters. During the

boot-up process endpoints register with a Resource Directory, which is hosted by the mobile operator or somewhere in the cloud. Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web applications for environment monitoring contact the RD, look up the endpoints capable of providing information about the environment using an appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server), and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look up for EPs deployed on the vehicles the application is responsible for.

3.5. 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.

3.6. 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 RD Lookup to discover and resolve links to the desired resources and endpoints. The Resource Directory service need not be coupled with the data intermediary service. Mapping of Resource Directories to data intermediaries may be many-to-many.

Metadata in web link formats like [RFC6690] are supplied by Resource Directories, which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links. External catalogs that are represented in other formats may be converted to common web linking formats for storage and access by Resource Directories. Since it is common practice for these to be URN encoded, simple and lossless structural transforms should generally be sufficient to store external metadata in Resource Directories.

The additional features of Resource Directory allow domains to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Resource groups may defined to allow batched reads from multiple resources.

4. Finding a Resource Directory

A (re-e)starting device may want to find one or more resource directories to make itself known with.

The device may be pre-configured to exercise specific mechanisms for finding the resource directory:

- It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an appropriate RD. (Instead of using an anycast address, a multicast address can also be preconfigured. The RD directory servers then need to configure one of their interfaces with this multicast address.)

- It may be configured with a DNS name for the RD and a resource-record type to look up under this name; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.

- It may be configured to use a service discovery mechanism such as DNS-SD [RFC6763]. The present specification suggests configuring the service with name rd._sub._coap._udp, preferably within the domain of the querying nodes.

For cases where the device is not specifically configured with a way to find a resource directory, the network may want to provide a suitable default.

- If the address configuration of the network is performed via SLAAC, this is provided by the RDAO option Section 4.1.

- If the address configuration of the network is performed via DHCP, this could be provided via a DHCP option (no such option is defined at the time of writing).

Finally, if neither the device nor the network offer any specific configuration, the device may want to employ heuristics to find a suitable resource directory.

The present specification does not fully define these heuristics, but suggests a number of candidates:

- In a 6LoWPAN, just assume the Edge Router (6LBR) can act as a resource directory (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a Unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".

- In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[MCD1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

As some of the RD addresses obtained by the methods listed here are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

If multiple candidate addresses are discovered, the device may pick any of them initially, unless the discovery method indicates a more precise selection scheme.

4.1. Resource Directory Address Option (RDAO)

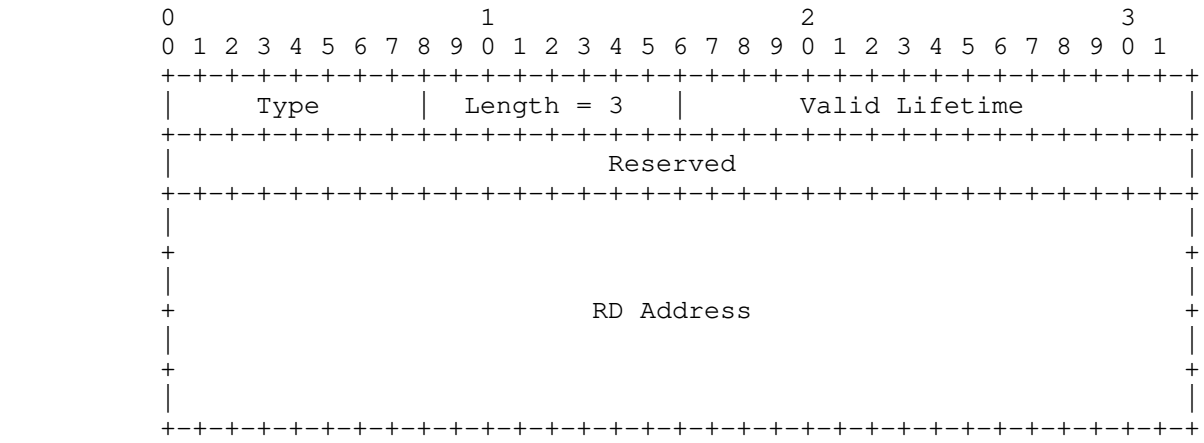
The Resource Directory Address Option (RDAO) using IPv6 neighbor Discovery (ND) carries information about the address of the Resource Directory (RD). This information is needed when endpoints cannot discover the Resource Directory with a link-local multicast address because the endpoint and the RD are separated by a border Router (6LBR). In many circumstances the availability of DHCP cannot be

guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many resource directory addresses.

The lifetime 0x0 means that the RD address is invalid and to be removed.

The RDAO format is:



Fields:

Type: 38

Length: 8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3.

Valid Lifetime: 16-bit unsigned integer. The length of time in units of 60 seconds (relative to the time the packet is received) that this Resource Directory address is valid. A value of all zero bits (0x0) indicates that this Resource Directory address is not valid anymore.

Reserved: This field is unused. It MUST be initialized to zero by the sender and MUST be ignored by the receiver.

RD Address: IPv6 address of the RD.

Figure 5: Resource Directory Address Option

5. Resource Directory

This section defines the required set of REST interfaces between a Resource Directory (RD) and endpoints. 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.

All operations on the contents of the Resource Directory MUST be atomic and idempotent.

A resource directory MAY make the information submitted to it 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.

5.1. Payload Content Formats

Resource Directory implementations using this specification MUST support the application/link-format content format (ct=40).

Resource Directories implementing this specification MAY support additional content formats.

Any additional content format supported by a Resource Directory implementing this specification MUST have an equivalent serialization in the application/link-format content format.

5.2. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690]. A complete set of RD discovery methods is described in Section 4.

Discovery of the RD registration URI path is performed by sending either a multicast or unicast GET request to "/.well-known/core" and including a Resource Type (rt) parameter [RFC6690] with the value "core.rd" in the query string. Likewise, a Resource Type parameter value of "core.rd-lookup*" is used to discover the URIs for RD Lookup operations, and "core.rd-group" is used to discover the URI path for RD Group operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI of the RD function returned and the corresponding Resource Type. When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network.

A Resource Directory MAY provide hints about the content-formats it supports in the links it exposes or registers, using the "ct" link attribute, as shown in the example below. Clients MAY use these

hints to select alternate content-formats for interaction with the Resource Directory.

HTTP does not support multicast and consequently only unicast discovery can be supported using HTTP. Links to Resource Directories MAY be registered in other Resource Directories. The well-known entry points SHOULD be provided to enable the bootstrapping of unicast discovery.

An implementation of this resource directory specification MUST support query filtering for the rt parameter as defined in [RFC6690].

While the link targets in this discovery step are often expressed in path-absolute form, this is not a requirement. Clients SHOULD therefore accept URIs of all schemes they support, both in absolute and relative forms, and not limit the set of discovered URIs to those hosted at the address used for URI discovery.

The URI Discovery operation can yield multiple URIs of a given resource type. The client can use any of the discovered addresses initially.

The discovery request interface is specified as follows:

Interaction: EP -> RD

Method: GET

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

URI Template Variables:

rt := Resource Type (optional). MAY contain one of the values "core.rd", "core.rd-lookup*", "core.rd-lookup-res", "core.rd-lookup-ep", "core.rd-lookup-gp", "core.rd-group" or "core.rd*"

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

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

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

The following response codes are defined for this interface:

Success: 2.05 "Content" 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: 4.00 "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.

HTTP support : YES (Unicast only)

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource is, in this example, at /rd, and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that it is up to the RD to choose its RD resource paths.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd";ct=40,
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct=40,
</rd-lookup/res>;rt="core.rd-lookup-res";ct=40,
</rd-lookup/gp>;rt="core.rd-lookup-gp";ct=40,
</rd-group>;rt="core.rd-group";ct=40

Figure 6: Example discovery exchange

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as the the CBOR and JSON representation of link format. The RD resource paths /rd, /rd-lookup, and /rd-group are example values. The server in this example also indicates that it is capable of providing observation on resource lookups.

[The RFC editor is asked to replace these and later occurrences of TBD64 and TBD504 with the numeric ID values assigned by IANA to application/link-format+cbor and application/link-format+json, respectively, as they are defined in I-D.ietf-core-links-json.]

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd";ct="40 65225",
</rd-lookup/res>;rt="core.rd-lookup-res";ct="40 TBD64 TBD504";obs,
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct="40 TBD64 TBD504",
</rd-lookup/gp>;rt="core.rd-lookup-gp";ct=40 TBD64 TBD504",
</rd-group>;rt="core.rd-group";ct="40 TBD64 TBD504"

From a management and maintenance perspective, it is necessary to identify the components that constitute the server. The identification refers to information about for example client-server incompatibilities, supported features, required updates and other aspects. The URI discovery address, a described in section 4 of [RFC6690] can be used to find the identification.

It would typically be stored in an implementation information link (as described in [I-D.bormann-t2trg-rel-impl]):

```
Req: GET /.well-known/core?rel=impl-info

Res: 2.05 Content
<http://software.example.com/shiny-resource-directory/1.0beta1>;
    rel="impl-info"
```

Note that depending on the particular server's architecture, such a link could be anchored at the server's root, at the discovery site (as in this example) or at individual RD components. The latter is to be expected when different applications are run on the same server.

5.3. Registration

After discovering the location of an RD, an endpoint MAY register its resources using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690], JSON CoRE Link Format (application/link-format+json), or CBOR CoRE Link Format (application/link-format+cbor) [I-D.ietf-core-links-json], along with query parameters indicating the name of the endpoint, and optionally the domain and the lifetime of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. An endpoint MUST use that location when refreshing registrations using this interface. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The endpoint is responsible for refreshing the registration resource within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d does not create multiple registration resources. A new registration resource may be created at any time to supersede an existing registration, replacing the registration parameters and links.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or

contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. The Base URI against which they are resolved is the context of the registration, which is provided either explicitly in the "con" parameter or constructed implicitly from the requester's network address.

Documents in [RFC6690] Link Format SHOULD NOT contain links in which resolving the target literal against the base URI gives a different result than resolving it against the resolved anchor; this is to avoid the ambiguities described in Appendix A.4. * Entries in which there is no anchor attribute, * entries in which the target is an absolute reference and * entries in which both the target and the anchor start with a slash ("/")

never cause that kind of ambiguity.

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: {+rd}{?ep,d,lt,con,extra-attrs*}

URI Template Variables:

rd := RD registration URI (mandatory). This is the location of the RD, as obtained from discovery.

ep := Endpoint name (mostly mandatory). The endpoint name is an identifier that MUST be unique within a domain. The maximum length of this parameter is 63 bytes. If the RD is configured to recognize the endpoint (eg. based on its security context), the endpoint can ignore the endpoint name, and assign one based on a se of configuration parameter values.

d := Domain (optional). The domain to which this endpoint belongs. The maximum length of this parameter is 63 bytes. When this parameter is not present, the RD MAY associate the endpoint with a configured default domain or leave it empty.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included in the initial registration, a default value of 86400 (24 hours) SHOULD be assumed.

con := Context (optional). This parameter sets the Default Base URI under which the request's links are to be interpreted. The specified URI MUST NOT have a path component of its own, but MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore of the shape "scheme://authority" for HTTP and CoAP URIs. In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. This parameter is mandatory when the directory is filled by a third party such as an commissioning tool. If the endpoint uses an ephemeral port to register with, it MUST include the con parameter in the registration to provide a valid network path. If the endpoint which is located behind a NAT gateway is registering with a Resource Directory which is on the network service side of the NAT gateway, the endpoint MUST use a persistent port for the outgoing registration in order to provide the NAT gateway with a valid network address for replies and incoming requests.

extra-attrs := Additional registration attributes (optional). The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

Content-Format: application/link-format

Content-Format: application/link-format+json

Content-Format: application/link-format+cbor

The following response codes are defined for this interface:

Success: 2.01 "Created" or 201 "Created". The Location header option MUST be included in the response when a new registration resource is created. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links. A registration with an already registered ep and d value pair responds with the same success code and Location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

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

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

If the registration fails with a Service Unavailable response and a Max-Age option or Retry-After header, the client SHOULD retry the operation after the time indicated. If the registration fails in another way, including request timeouts, or if the Service Unavailable error persists after several retries, or indicates a longer time than the endpoint is willing to wait, it SHOULD pick another registration URI from the "URI Discovery" step and if there is only one or the list is exhausted, pick other choices from the "Finding a Resource Directory" step. Care has to be taken to consider the freshness of results obtained earlier, eg. of the result of a "/.well-known/core" response, the lifetime of an RDAO option and of DNS responses. Any rate limits and persistent errors from the "Finding a Resource Directory" step must be considered for the whole registration time, not only for a single operation.

The following example shows an endpoint with the name "node1" registering two resources to an RD using this interface. The location "/rd" is an example RD location discovered in a request similar to Figure 6.

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor";
      anchor="coap://spurious.example.com:5683",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

Res: 2.01 Created
Location: /rd/4521
```

Figure 7: Example registration payload

A Resource Directory may optionally support HTTP. Here is an example of almost the same registration operation above, when done using HTTP and the JSON Link Format.

```
Req: POST /rd?ep=node1&con=http://[2001:db8:1::1] HTTP/1.1
Host : example.com
Content-Type: application/link-format+json
Payload:
[
{"href": "/sensors/temp", "ct": "41", "rt": "temperature-c", "if": "sensor",
 "anchor": "coap://spurious.example.com:5683"},
{"href": "/sensors/light", "ct": "41", "rt": "light-lux", "if": "sensor"}
]

Res: 201 Created
Location: /rd/4521
```

5.3.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to a RD as described in Section 5.3. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification MUST implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the 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 links in that document are subject to the same limitations as the payload of a registration (no relative target references when anchor is present).

The endpoint then finds one or more addresses of the directory server as described in Section 4.

An endpoint finally asks the selected directory server to probe it for resources and publish them as follows:

The endpoint sends (and regularly refreshes with) a POST request to the "/.well-known/core" URI of the directory server of choice. The body of the POST request is empty, which triggers the resource directory server to perform GET requests at the requesting server's default discovery URI to obtain the link-format payload to register.

The endpoint includes the same registration parameters in the POST request as it would per Section 5.3. The context of the registration is taken from the requesting server's URI.

The endpoints MUST be deleted after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

The following example shows an endpoint using Simple Registration, by simply sending an empty POST to a resource directory.

```
Req:(to RD server from [2001:db8:2::1])
POST /.well-known/core?lt=6000&ep=node1
Content-Format: 40
No payload

Res: 2.04 Changed

(later)

Req: (from RD server to [2001:db8:2::1])
GET /.well-known/core
Accept: 40

Res: 2.05 Content
Payload:
</sen/temp>
```

5.3.2. Third-party registration

For some applications, even Simple Registration may be too taxing for some very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the Resource Directory can be filled by a third device, called a commissioning tool. The commissioning 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 described in Section 5.3.

It should be noted that the value of the con parameter applies to all the links of the registration and has consequences for the anchor value of the individual links as exemplified in Appendix A. An eventual (currently non-existing) con attribute of the link is not affected by the value of con parameter in the registration.

5.4. Operations on the Registration Resource

After the initial registration, an endpoint should retain the returned location of the Registration Resource for further operations, including refreshing the registration in order to extend the lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this endpoint. The RD SHOULD continue to provide access to the Registration Resource after a registration time-out

occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of garbage collection and remove it from any group it belongs to. If the Registration Resource is removed, the endpoint will need to re-register.

The Registration Resource may also be used to inspect the registration resource using GET, update the registration, or cancel the registration using DELETE.

These operations are described in this section.

5.4.1. Registration 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 registration resource returned by the initial registration operation.

An update MAY update the lifetime- or the context- registration parameters "lt", "con" as in Section 5.3. Parameters that are not being changed SHOULD NOT be included in an update. Adding parameters that have not changed increases the size of the message but does not have any other implications. Parameters MUST be included as query parameters in an update operation as in Section 5.3.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the context of the registration is changed in an update explicitly or implicitly, relative references submitted in the original registration or later updates are resolved anew against the new context (like in the original registration).

The registration update operation only describes the use of POST with an empty payload. Future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration (see Section 5.4.4).

The update registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: {+location}{?lt,con,extra-attrs*}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 86400) SHOULD be used.

con := Context (optional). This parameter updates the context established in the original registration to a new value. If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the links of the registration, following the same restrictions as in the registration. If the parameter is not set and was set explicitly before, the previous context value is kept unmodified. If the parameter is not set and was not set explicitly before either, the source address and source port of the update request are stored as the context.

extra-attrs := Additional registration attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Content-Format: none (no payload)

The following response codes are defined for this interface:

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

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

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

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

If the registration update fails with a "Service Unavailable" response and a Max-Age option or Retry-After header, the client SHOULD retry the operation after the time indicated. If the

registration fails in another way, including request timeouts, or if the time indicated excedes the remaining lifetime, the client SHOULD attempt registration again.

The following example shows an endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

The following example shows an endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the client set the following values:

- endpoint name (ep)=endpoint1
- lifetime (lt)=500
- context (con)=coap://local-proxy-old.example.com:5683
- payload of Figure 7

The initial state of the Resource Directory is reflected in the following request:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
<coap://local-proxy-old.example.com:5683/sensors/temp>;ct=41;rt="temperature";
    anchor="coap://spurious.example.com:5683",
<coap://local-proxy-old.example.com:5683/sensors/light>;ct=41;rt="light-lux";
    if="sensor";anchor="coap://local-proxy-old.example.com:5683"
```

The following example shows an EP changing the context to "coaps://new.example.com:5684":

Req: POST /rd/4521?con=coaps://new.example.com:5684

Res: 2.04 Changed

The consecutive query returns:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
<coaps://new.example.com:5684/sensors/temp>;ct=41;rt="temperature";
    anchor="coap://spurious.example.com:5683",
<coaps://new.example.com:5684/sensors/light>;ct=41;rt="light-lux";if="sensor";
    anchor="coaps://new.example.com:5684",
```

5.4.2. Registration 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.

Removed endpoints are implicitly removed from the groups to which they belong.

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

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

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

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

5.4.3. Read Endpoint Links

Some endpoints may wish to manage their links as a collection, and may need to read the current set of links stored in the registration resource, in order to determine link maintenance operations.

One or more links MAY be selected by using query filtering as specified in [RFC6690] Section 4.1

If no links are selected, the Resource Directory SHOULD return an empty payload.

The read request interface is specified as follows:

Interaction: EP -> RD

Method: GET

URI Template: {+location}{?href,rel,rt,if,ct}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

href,rel,rt,if,ct := link relations and attributes specified in the query in order to select particular links based on their relations and attributes. "href" denotes the URI target of the link. See [RFC6690] Sec. 4.1

The following response codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" upon success with an "application/link-format", "application/link-format+cbor", or "application/link-format+json" payload.

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

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

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples show successful read of the endpoint links from the RD, with example location value /rd/4521 and example registration payload of Figure 7.

Req: GET /rd/4521

Res: 2.01 Content
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor";
anchor="coap://spurious.example.com:5683",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

5.4.4. Update Endpoint Links

An iPATCH (or PATCH) update [RFC8132] adds, removes or changes links of a registration by including link update information in the payload of the update with a media type that still needs to be defined.

6. RD Groups

This section defines the REST API for the creation, management, and lookup of endpoints for group operations. Similar to endpoint registration entries in the RD, groups 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 commissioning tool (CT) 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. This specification does not require that the endpoints belong to the same domain as the group, but a Resource Directory implementation can impose requirements on the domains of groups and endpoints depending on its configuration.

The registration message is a list of links to registration resources of the endpoints that belong to that group. The registration resources MAY be located on different hosts than the group hosting RD. In that case the endpoint link points to the registration

resource on the other RD. The commissioning tool SHOULD NOT attempt to enter a foreign registration in a group unless it found it in the group RD's lookup results, or has other reasons to assume that the foreign registration will be accepted.

The commissioning tool SHOULD not send any target attributes with the links to the registration resources, and the resource directory SHOULD reject registrations that contain links with unprocessable attributes.

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: CT -> RD

Method: POST

URI Template: {+rd-group}{?gp,d,con}

URI Template Variables:

rd-group := RD Group URI (mandatory). This is the location of the RD Group REST API.

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. When this parameter is not present, the RD MAY associate the group with a configured default domain or leave it empty.

con := Context (optional). This parameter sets the scheme, address and port of the multicast address associated with the group. When con is used, scheme and host are mandatory and port parameter is optional.

Content-Format: application/link-format

Content-Format: application/link-format+json

Content-Format: application/link-format+cbor

The following response codes are defined for this interface:

Success: 2.01 "Created" or 201 "Created". The Location header option MUST be returned in response to a successful group CREATE operation. This Location MUST be a stable identifier generated by the RD as it is used for delete operations of the group resource.

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

Failure: 4.04 "Not Found" or 404 "Not Found". An Endpoint is not registered in the RD (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following example shows an EP registering a group with the name "lights" which has two endpoints. The RD group path /rd-group is an example RD location discovered in a request similar to Figure 6.

```
Req: POST coap://rd.example.com/rd-group?gp=lights
                                    &con=coap://[ff35:30:2001:db8::1]
Content-Format: 40
Payload:
<coap://other-rd/rd/4521>,
</rd/4522>

Res: 2.01 Created
Location: /rd-group/12
```

A relative href value denotes the path to the registration resource of the Endpoint. When pointing to a registration resource on a different RD, the href value is an absolute URI.

6.2. Group Removal

A group can be removed simply by sending a removal message to the location of the group registration resource which was returned when initially 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: CT -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables:

location := This is the path of the group resource returned by the RD as a result of a successful group registration.

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

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

Failure: 4.04 "Not Found" or 404 "Not Found". Group does not exist.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples shows successful removal of the group from the RD with the example location value /rd-group/12.

Req: DELETE /rd-group/12

Res: 2.02 Deleted

7. RD Lookup

To discover the resources registered with the RD, a lookup interface must be provided. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. Atom or HTML Link) or using more advanced interfaces (e.g. supporting context or semantic based lookup).

RD Lookup allows lookups for groups, endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, a group lookup MUST return a list of groups, an endpoint lookup MUST return a list of endpoints and a resource lookup MUST return a list of links to resources.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

Lookup Type	Resource Type	Mandatory
Resource	core.rd-lookup-res	Mandatory
Endpoint	core.rd-lookup-ep	Mandatory
Group	core.rd-lookup-gp	Optional

Table 1: Lookup Types

7.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD if they were accessed on the endpoint itself. The links and link parameters returned are equal to the submitted, except that the target and anchor references are fully resolved.

Links that did not have an anchor attribute are therefore returned with the (explicitly or implicitly set) context URI of the registration as the anchor. Links whose href or anchor was submitted as an absolute URI are returned with respective attributes unmodified.

Above rules allow the client to interpret the response as links without any further knowledge of what the RD does. The Resource Directory MAY replace the contexts with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

7.2. Endpoint and group lookup

Endpoint and group lookups result in links to registration resources and group resources, respectively. Endpoint registration resources are annotated with their endpoint names (ep), domains (d, if present), context (con) and lifetime (lt, if present). Additional endpoint attributes are added as link attributes to their endpoint link unless their specification says otherwise. Group resources are annotated with their group names (gp), domain (d, if present) and multicast address (con, if present).

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

A Resource Directory can report endpoints or groups in lookup that are not hosted at the same address. While the setup and management of such a distributed system is out of scope for this document,

lookup clients MUST be prepared to see arbitrary URIs as registration or group resources in the results.

For groups, a Resource Directory as specified here does not provide a lookup mechanism for the resources that can be accessed on a group's multicast address (ie. no lookup will return links like "<coap://[ff35:30:2001:db8::1]/light>;..." for a group registered with "con=coap://[ff35...]"). Such an additional lookup interface could be specified in an extension document.

7.3. Lookup filtering

Using the Accept Option, the requester can control whether the returned list is returned in CoRE Link Format ("application/link-format", default) or its alternate content-formats ("application/link-format+json" or "application/link-format+cbor").

The page and count parameters are used to obtain lookup results in specified increments using pagination, where count specifies how many links to return and page specifies which subset of links organized in sequential pages, each containing 'count' links, starting with link zero and page zero. Thus, specifying count of 10 and page of 0 will return the first 10 links in the result set (links 0-9). Count = 10 and page = 1 will return the next 'page' containing links 10-19, and so on.

Multiple search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned. The Resource Directory MUST support matching with multiple search criteria.

A link matches a search criterion if it has an attribute of the same name and the same value, allowing for a trailing "*" wildcard operator as in Section 4.1 of [RFC6690]. Attributes that are defined as "link-type" match if the search value matches any of their values (see Section 4.1 of [RFC6690]; eg. "?if=core.s" matches ";if="abc core.s";"). A link also matches a search criterion if the link that would be produced for any of its containing entities would match the criterion, or an entity contained in it would: A search criterion matches an endpoint if it matches the endpoint itself, any of the groups it is contained in or any resource it contains. A search criterion matches a resource if it matches the resource itself, the resource's endpoint, or any of the endpoint's groups.

Note that "href" is also a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it, or any group resource that endpoint is contained in.

Clients that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

When resource observation is used, every time the set of matching links changes, or the content of a matching link changes, the RD sends a notification with the matching link set. The notification contains the successful current response to the given request, especially with respect to representing zero matching links (see "Success" item below).

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables:

type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 5.2.

search := Search criteria for limiting the number of results (optional).

page := Page (optional). Parameter can not be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

Content-Format: application/link-format (optional)

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

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

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. The payload can contain zero links (which is an empty payload, "80" (hex) or "[]" in the respective content format), indicating that no entities matched the request.

Failure: No error response to a multicast request.

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

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

7.4. Lookup examples

The examples in this section assume the existence of CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example resource look-up locations discovered in Figure 6:

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

Res: 2.05 Content
<coap://[2001:db8:3::123]:61616/temp>;rt="temperature";anchor="coap://[2001:db8:
3::123]:61616"
```

The same lookup using the CBOR Link Format media type:

```
Req: GET /rd-lookup/res?rt=temperature
Accept: TBD64

Res: 2.05 Content
Content-Format: TBD64
Payload in Hex notation:
81A3017823636F61703A2F2F5B323030313A6462383A333A3A3132335D3A36313631362F
74656D7003781E636F61703A2F2F5B323030313A6462383A333A3A3132335D3A36313631
36096B74656D7065726174757265
Decoded payload:
[{1: "coap://[2001:db8:3::123]:61616/temp", 9: "temperature",
3: "coap://[2001:db8:3::123]:61616"}]
```

A client that wants to be notified of new resources as they show up can use observation:

```
Req: GET /rd-lookup/res?rt=light
Observe: 0

Res: 2.05 Content
Observe: 23
Payload: empty

(at a later point in time)

Res: 2.05 Content
Observe: 24
Payload:
<coap://[2001:db8:3::124]/west>;rt="light";
    anchor="coap://[2001:db8:3::124]",
<coap://[2001:db8:3::124]/south>;rt="light";
    anchor="coap://[2001:db8:3::124]",
<coap://[2001:db8:3::124]/east>;rt="light";
    anchor="coap://[2001:db8:3::124]"
```

The following example shows a client performing an endpoint type (et) lookup with the value oic.d.sensor (which is currently a registered rt value):

```
Req: GET /rd-lookup/ep?et=oic.d.sensor

Res: 2.05 Content
</rd/1234>;con="coap://[2001:db8:3::127]:61616";ep="node5";
et="oic.d.sensor";ct="40";lt="600",
</rd/4521>;con="coap://[2001:db8:3::129]:61616";ep="node7";
et="oic.d.sensor";ct="40";lt="600";d="floor-3"
```

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

```
Req: GET /rd-lookup/gp

Res: 2.05 Content
</rd-group/1>;gp="lights1";d="example.com";con="coap://[ff35:30:2001:db8::1]",
</rd-group/2>;gp="lights2";d="example.com";con="coap://[ff35:30:2001:db8::2]"
```

The following example shows a client performing a lookup for all endpoints in a particular group, with one endpoint hosted by another RD:

```
Req: GET /rd-lookup/ep?gp=lights1

Res: 2.05 Content
<coap://[other-rd]/rd/abcd>;con="coap://[2001:db8:3::123]:61616";
anchor="coap://[other-rd]";ep="node1";et="oic.d.sensor";ct="40";lt="600",
</rd/efgh>;con="coap://[2001:db8:3::124]:61616";
ep="node2";et="oic.d.sensor";ct="40";lt="600"
```

The following example shows a client performing a lookup for all groups the endpoint "node1" belongs to:

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

Res: 2.05 Content
</rd-group/1>;gp="lights1"
```

The following example shows a client performing a paginated resource lookup

```
Req: GET /rd-lookup/res?page=0&count=5

Res: 2.05 Content
<coap://[2001:db8:3::123]:61616/res/0>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/1>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/2>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/3>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/4>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616"

Req: GET /rd-lookup/res?page=1&count=5

Res: 2.05 Content
<coap://[2001:db8:3::123]:61616/res/5>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/6>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/7>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/8>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/9>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616"
```

The following example shows a client performing a lookup of all resources from endpoints of all endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th request of section 5 of [RFC6690].

It demonstrates how absolute link targets stay unmodified, while relative ones are resolved:

```
Req: GET /rd-lookup/res?et=oic.d.sensor

<coap://sensor1.example.com/sensors>;ct=40;title="Sensor Index";
    anchor="coap://sensor1.example.com",
<coap://sensor1.example.com/sensors/temp>;rt="temperature-c";if="sensor";
    anchor="coap://sensor1.example.com",
<coap://sensor1.example.com/sensors/light>;rt="light-lux";if="sensor";
    anchor="coap://sensor1.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
    anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor1.example.com/t>;rel="alternate";
    anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor2.example.com/sensors>;ct=40;title="Sensor Index";
    anchor="coap://sensor2.example.com",
<coap://sensor2.example.com/sensors/temp>;rt="temperature-c";if="sensor";
    anchor="coap://sensor2.example.com",
<coap://sensor2.example.com/sensors/light>;rt="light-lux";if="sensor";
    anchor="coap://sensor2.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
    anchor="coap://sensor2.example.com/sensors/temp",
<coap://sensor2.example.com/t>;rel="alternate";
    anchor="coap://sensor2.example.com/sensors/temp"
```

8. 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.

8.1. Endpoint Identification and Authentication

An Endpoint is determined to be unique within (the domain of) 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.

Consider te following threat: two devices A and B are managed by a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the DTLS exchange. Then, it puts the endpoint name of device B. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Therfore, Endpoints MUST include the Endpoint identifier in the message, and this identifier MUST be checked by a resource directory to match the Endpoint identifier included in the Registration message.

8.2. Access Control

Access control SHOULD be performed separately for the RD registration, Lookup, and group API paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the domain, endpoint or resource level.

8.3. 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. There is also a danger that NTP Servers could become implicated in denial-of-service (DoS) attacks since they run on unprotected UDP, there is no return routability check, and they 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.

9. IANA Considerations

9.1. Resource Types

"core.rd", "core.rd-group", "core.rd-lookup-ep", "core.rd-lookup-res", and "core.rd-lookup-gp" resource types need to be registered with the resource type registry defined by [RFC6690].

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the subregistry "IPv6 Neighbor Discovery Option Formats":

- Resource Directory address Option (38)

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include

- the human readable name of the parameter,
- the short name as used in query parameters or link attributes,
- indication of whether it can be passed as a query parameter at registration of endpoints or groups, as a query parameter in lookups, or be expressed as a link attribute,
- validity requirements if any, and
- a description.

The query parameter MUST be both a valid URI query key [RFC3986] and a parmname as used in [RFC5988].

The description must give details on which registrations they apply to (Endpoint, group registrations or both? Can they be updated?), and how they are to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at

registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

Full name	Short	Validity	Use	Description
Endpoint Name	ep		RLA	Name of the endpoint, max 63 bytes
Lifetime	lt	60-4294967295	RLA	Lifetime of the registration in seconds
Domain	d		RLA	Domain to which this endpoint belongs
Context	con	URI	RLA	The scheme, address and port and path at which this server is available
Group Name	gp		RLA	Name of a group in the RD
Page	page	Integer	L	Used for pagination
Count	count	Integer	L	Used for pagination
Endpoint Type	et		RLA	Semantic name of the endpoint (see Section 9.4)

Table 2: RD Parameters

(Short: Short name used in query parameters or link attributes. Use: R = used at registration, L = used at lookup, A = expressed in link attribute

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (Eg. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used link attributes (For example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but

would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] link attribute). It is expected that the registry will receive between 5 and 50 registrations in total over the next years.

9.3.1. Full description of the "Endpoint Type" Registration Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type subregistry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, Resource Directory implementations automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called '"Endpoint Type" (et=) RD Parameter values'. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

- The values MUST be related to the purpose described in Section 9.3.1.
- The registered values MUST conform to the ABNF reg-rel-type definition of [RFC6690] and MUST NOT be a URI.
- It is recommended to use the period "." character for segmentation.

The registry is initially empty.

9.5. Multicast Address Registration

IANA has assigned the following multicast addresses for use by CoAP nodes:

IPv4 - "all CoRE resource directories" address, from the "IPv4 Multicast Address Space Registry" equal to "All CoAP Nodes", 224.0.1.187. As the address is used for discovery that may span beyond a single network, it has come from the Internetwork Control Block (224.0.1.x, RFC 5771).

IPv6 - "all CoRE resource directories" address MCD1 (uggestions FF0X::FE), from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LWM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the Resource Directory (RD) with a CoAP interface to facilitate the installation and start up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative.

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct

functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infra structure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and sensor shown in Table 3 below:

Name	IPv6 address
luminary1	2001:db8:4::1
luminary2	2001:db8:4::2
Presence sensor	2001:db8:4::3
Resource directory	2001:db8:4::ff

Table 3: interface SLAAC addresses

In Section 10.1.2 the use of resource directory during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have rt: light, and the presence sensor has rt: p-sensor. The endpoints have names which are relevant to the light installation manager. In this case luminary1, luminary2, and the presence sensor are located in room 2-4-015, where luminary1 is

located at the window and luminary2 and the presence sensor are located at the door. The endpoint names reflect this physical location. The middle, left and right lamps are accessed via path /light/middle, /light/left, and /light/right respectively. The identifiers relevant to the Resource Directory are shown in Table 4 below:

Name	endpoint	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	light
luminary1	lm_R2-4-015_wndw	/light/middle	light
luminary1	lm_R2-4-015_wndw	/light/right	light
luminary2	lm_R2-4-015_door	/light/left	light
luminary2	lm_R2-4-015_door	/light/middle	light
luminary2	lm_R2-4-015_door	/light/right	light
Presence sensor	ps_R2-4-015_door	/ps	p-sensor

Table 4: Resource Directory identifiers

It is assumed that the CT knows the RD's address, and has performed URI discovery on it that returned a response like the one in the Section 5.2 example.

The CT inserts the endpoints of the luminaries and the sensor in the RD using the Context parameter (con) to specify the interface address:

```
Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=lm_R2-4-015_wndw&con=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"

Res: 2.01 Created
Location: /rd/4521
```

```
Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=lm_R2-4-015_door&con=coap://[2001:db8:4::2]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"

Res: 2.01 Created
Location: /rd/4522

Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=ps_R2-4-015_door&con=coap://[2001:db8:4::3]d&d=R2-4-015
Payload:
</ps>;rt="p-sensor"

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 more awkward. The same domain name is communicated to the two luminaries and the presence sensor by the CT.

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 endpoints and the endpoint of the presence sensor are registered as members of the group.

```
Req: POST coap://[2001:db8:4::ff]/rd-group
?gp=grp_R2-4-015&con=coap://[ff05::1]
Payload:
</rd/4521>,
</rd/4522>,
</rd/4523>

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 endpoint with rt=light and d=R2-4-015. The RD returns all endpoints in the domain.

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
  ?d=R2-4-015&rt=light

Res: 2.05 Content
</rd/4521>;con="coap://[2001:db8:4::1]";
  ep="lm_R2-4-015_wndw",
</rd/4522>;con="coap://[2001:db8:4::2]";
   ep="lm_R2-4-015_door"
```

Knowing its own IPv6 address, the luminary discovers its endpoint name. With the endpoint name the luminary queries the RD for all groups to which the endpoint belongs.

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/gp
  ?ep=lm_R2-4-015_wndw

Res: 2.05 Content
</rd-group/501>;gp="grp_R2-4-015";con="coap://[ff05::1]"
```

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 coap://[2001:db8:4::1]/coap-group
Content-Format: application/coap-group+json
Payload:
{ "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.

10.2. 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(OMA Name Authority). 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, and applications are currently out of scope for LWM2M.

The location of the LWM2M Server and RD URI path is provided by the LWM2M Bootstrap process, so no dynamic discovery of the RD is used. LWM2M Servers and endpoints are not required to implement the /.well-known/core resource.

10.2.1. 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".

base-uri := Base URI for LWM2M resources or "undefined" for default (empty) base URI

object-id := OMNA (OMA Name Authority) 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 (OMA Name Authority) 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).

10.2.2. LWM2M Register Endpoint

LWM2M defines a registration interface based on the REST API, described in Section 5. The RD registration URI path of the LWM2M Resource Directory is specified to be "/rd".

LWM2M endpoints register object IDs, for example </1>, to indicate that a particular object type is supported, and register object instances, for example </1/0>, to indicate that a particular instance of that object type exists.

Resources within the LWM2M object instance are not registered with the RD, but may be discovered by reading the resource links from the object instance using GET with a CoAP Content-Format of application/link-format. Resources may also be read as a structured object by

performing a GET to the object instance with a Content-Format of senml+json.

When an LWM2M object or instance is registered, this indicates to the LWM2M server that the object and its resources are available for management and service enablement (REST API) operations.

LWM2M endpoints may use the following RD registration parameters as defined in Table 2 :

ep - Endpoint Name
lt - registration lifetime

Endpoint Name, Lifetime, and LWM2M Version are mandatory parameters for the register operation, all other registration parameters are optional.

Additional optional LWM2M registration parameters are defined:

Name	Query	Validity	Description
Binding Mode	b	{"U",UQ","S","SQ","US","UQS"}	Available Protocols
LWM2M Version	ver	1.0	Spec Version
SMS Number	sms		MSISDN

Table 5: LWM2M Additional Registration Parameters

The following RD registration parameters are not currently specified for use in LWM2M:

et - Endpoint Type
con - Context

The endpoint registration must include a payload containing links to all supported objects and existing object instances, optionally including the appropriate link-format relations.

Here is an example LWM2M registration payload:

</1>,</1/0>,</3/0>,</5>

This link format payload indicates that object ID 1 (LWM2M Server Object) is supported, with a single instance 0 existing, object ID 3 (LWM2M Device object) is supported, with a single instance 0 existing, and object 5 (LWM2M Firmware Object) is supported, with no existing instances.

10.2.3. LWM2M Update Endpoint Registration

The LwM2M update is really very similar to the registration update as described in Section 5.4.1, with the only difference that there are more parameters defined and available. All the parameters listed in that section are also available with the initial registration but are all optional:

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.

10.2.4. 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 5.4.2.

11. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, and Jan Newmarch 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.

12. Changelog

changes from -12 to -13

- Added "all resource directory" nodes MC address
- Clarified observation behavior
- version identification

- example rt= and et= values
- domain from figure 2
- more explanatory text
- endpoints of a groups hosted by different RD
- resolve RFC6690-vs-8288 resolution ambiguities:
  - require registered links not to be relative when using anchor
  - return absolute URIs in resource lookup

changes from -11 to -12

- added Content Model section, including ER diagram
- removed domain lookup interface; domains are now plain attributes of groups and endpoints
- updated chapter "Finding a Resource Directory"; now distinguishes configuration-provided, network-provided and heuristic sources
- improved text on: atomicity, idempotency, lookup with multiple parameters, endpoint removal, simple registration
- updated LWM2M description
- clarified where relative references are resolved, and how context and anchor interact
- new appendix on the interaction with RFCs 6690, 5988 and 3986
- lookup interface: group and endpoint lookup return group and registration resources as link targets
- lookup interface: search parameters work the same across all entities
- removed all methods that modify links in an existing registration (POST with payload, PATCH and iPATCH)
- removed plurality definition (was only needed for link modification)
- enhanced IANA registry text

- state that lookup resources can be observable
- More examples and improved text

changes from -09 to -10

- removed "ins" and "exp" link-format extensions.
- removed all text concerning DNS-SD.
- removed inconsistency in RDAO text.
- suggestions taken over from various sources
- replaced "Function Set" with "REST API", "base URI", "base path"
- moved simple registration to registration section

changes from -08 to -09

- clarified the "example use" of the base RD resource values /rd, /rd-lookup, and /rd-group.
- changed "ins" ABNF notation.
- various editorial improvements, including in examples
- clarifications for RDAO

changes from -07 to -08

- removed link target value returned from domain and group lookup types
- Maximum length of domain parameter 63 bytes for consistency with group
- removed option for simple POST of link data, don't require a .well-known/core resource to accept POST data and handle it in a special way; we already have /rd for that
- add IPv6 ND Option for discovery of an RD
- clarify group configuration section 6.1 that endpoints must be registered before including them in a group
- removed all superfluous client-server diagrams

- simplified lighting example
- introduced Commissioning Tool
- RD-Look-up text is extended.

changes from -06 to -07

- added text in the discovery section to allow content format hints to be exposed in the discovery link attributes
- editorial updates to section 9
- update author information
- minor text corrections

Changes from -05 to -06

- added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

- added Update Endpoint Links using PATCH
- http access made explicit in interface specification
- Added http examples

Changes from -03 to -04:

- Added http response codes
- Clarified endpoint name usage
- Add application/link-format+cbor content-format

Changes from -02 to -03:

- Added an example for lighting and DNS integration
- Added an example for RD use in OMA LWM2M
- Added Read Links operation for link inspection by endpoints
- Expanded DNS-SD section

- Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

- Added a catalogue use case.
- Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.
- Additional examples section added for more complex use cases.
- New DNS-SD mapping section.
- Added text on endpoint identification and authentication.
- Error code 4.04 added to Registration Update and Delete requests.
- Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- Removed the ETag validation feature.
- Place holder for the DNS-SD mapping section.
- Explicitly disabled GET or POST on returned Location.
- New registry for RD parameters.
- Added support for the JSON Link Format.
- Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- Updated the version and date.

Changes from -04 to -05:

- Restricted Update to parameter updates.
- Added pagination support for the Lookup interface.
- Minor editing, bug fixes and reference updates.
- Added group support.

- Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- Added the ins= parameter back for the DNS-SD mapping.
- Integrated the Simple Directory Discovery from Carsten.
- Editorial improvements.
- Fixed the use of ETags.
- Fixed tickets 383 and 372

Changes from -02 to -03:

- Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- Updated REST interface descriptions to use RFC6570 URI Template format.
- Introduced an improved RD Lookup design as its own function set.
- Improved the security considerations section.
- Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

- Added a terminology section.
- Changed the inclusion of an ETag in registration or update to a MAY.
- Added the concept of an RD Domain and a registration parameter for it.
- Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

13. References

13.1. Normative References

[I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", draft-ietf-core-links-json-10 (work in progress), February 2018.

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

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

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

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

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

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

13.2. Informative References

[ER] Chen, P., "The entity-relationship model---toward a unified view of data", ACM Transactions on Database Systems Vol. 1, pp. 9-36, DOI 10.1145/320434.320440, March 1976.

[I-D.arkko-core-dev-urn] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-05 (work in progress), October 2017.

[I-D.bormann-t2trg-rel-impl] Bormann, C., "impl-info: A link relation type for disclosing implementation information", draft-bormann-t2trg-rel-impl-00 (work in progress), January 2018.

[I-D.silverajan-core-coap-protocol-negotiation] Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", draft-silverajan-core-coap-protocol-negotiation-07 (work in progress), October 2017.

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

[RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <https://www.rfc-editor.org/info/rfc6775>.

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

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

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

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

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

Appendix A. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC8288] which defines link headers, and [RFC7252] providing the transport). This appendix summarizes the mechanisms and semantics at play from an entry in ".well-known/core" to a resource lookup.

This text is primarily aimed at people entering the field of Constrained Restful Environments from applications that previously did not use web mechanisms.

A.1. A simple example

Let's start this example with a very simple host, "2001:db8:f0::1". A client that follows classical CoAP Discovery ([RFC7252] Section 7), sends the following multicast request to learn about neighbours supporting resources with resource-type "temperature".

The client sends a link-local multicast:

GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature

RES 2.05 Content
</temp>;rt=temperature;ct=0

where the response is sent by the server, "[2001:db8:f0::1]:5683".

While the client - on the practical or implementation side - can just go ahead and create a new request to "[2001:db8:f0::1]:5683" with Uri-Path: "temp", the full resolution steps without any shortcuts are:

A.1.1. Resolving the URIs

The client parses the single returned record. The link's target (sometimes called "href") is ""/temp"", which is a relative URI that needs resolving. As long as all involved links follow the

restrictions set forth for this document (see Appendix A.4), the base URI to resolve this against the requested URI.

The URI of the requested resource can be composed by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into ""coap://[2001:db8:f0::1]/.well-known/core"".

The record's target is resolved by replacing the path ""/.well-known/core"" from the Base URI (section 5.2 [RFC3986]) with the relative target URI ""/temp"" into ""coap://[2001:db8:f0::1]/temp"".

A.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content type is text/plain (ct=0).

A relation in a web link is a three-part statement that the context resource has a named relation to the target resource, like "_This page_ has _its table of contents_ at _/toc.html_". In [RFC6690] link-format documents, there is an implicit "host relation" specified with default parameter: rel="hosts".

In our example, the context of the link is the URI of the requested document itself. A full English expression of the "host relation" is:

'"coap://[2001:db8:f0::1]/.well-known/core" is hosting the resource "coap://[2001:db8:f0::1]/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'

A.2. A slightly more complex example

Omitting the "rt=temperature" filter, the discovery query would have given some more records in the payload:

```
</temp>;rt=temperature;ct=0,
</light>;rt=light-lux;ct=0,
</t>;anchor="/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;anchor="/sensors/temp";
    rel="describedby"
```

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the document's Base URI and is thus resolved to ""coap://[2001:db8:f0::1]/sensors/temp"". That is the context resource of the link, so the "rel" statement is not about

the target and the document Base URI any more, but about the target and that address.

Thus, the third record could be read as ""coap://[2001:db8:f0::1]/sensors/temp" has an alternate representation at "coap://[2001:db8:f0::1]/t"".

The fourth record can be read as ""coap://[2001:db8:f0::1]/sensors/temp" is described by "http://www.example.com/sensors/t123"".

A.3. Enter the Resource Directory

The resource directory tries to carry the semantics obtainable by classical CoAP discovery over to the resource lookup interface as faithfully as possible.

For the following queries, we will assume that the simple host has used Simple Registration to register at the resource directory that was announced to it, sending this request from its UDP port "[2001:db8:f0::1]:6553":

```
POST coap://[2001:db8:f01::ff]/.well-known/core?ep=simple-host1
```

The resource directory would have accepted the registration, and queried the simple host's ".well-known/core" by itself. As a result, the host is registered as an endpoint in the RD with the name "simple-host1". The registration is active for 86400 seconds, and the endpoint registration Base URI is ""coap://[2001:db8:f0::1]/"" because that is the address the registration was sent from (and no explicit "con=" was given).

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching "coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res", obtain "coap://[2001:db8:f0::ff]/rd-lookup/res" as the resource lookup endpoint, and issue a request to "coap://[2001:db8:f0::ff]/rd-lookup/res?rt=temperature" to receive the following data:

```
<coap://[2001:db8:f0::1]/temp>;rt=temperature;ct=0;
    anchor="coap://[2001:db8:f0::1]"
```

This is not _literally_ the same response that it would have received from a multicast request, but it would contain the (almost) same statement:

'"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/temp", which is of the resource type

"temperature" and can be accessed using the text/plain content format.'

(The difference is whether "/" or "/.well-known/core" hosts the resources, which is subject of ongoing discussion about RFC6690).

To complete the examples, the client could also query all resources hosted at the endpoint with the known endpoint name "simple-host1". A request to "coap://[2001:db8:f0::ff]/rd-lookup/res?ep=simple-host1" would return

```
<coap://[2001:db8:f0::1]/temp>;rt=temperature;ct=0;
    anchor="coap://[2001:db8:f0::1]",
<coap://[2001:db8:f0::1]/light>;rt=light-lux;ct=0;
    anchor="coap://[2001:db8:f0::1]",
<coap://[2001:db8:f0::1]/t>;
    anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;
    anchor="coap://[2001:db8:f0::1]/sensors/temp";rel="describedby"
```

All the target and anchor references are already in absolute form there, which don't need to be resolved any further.

Had the simple host registered with an explicit context (eg. "?ep=simple-host1&con=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

```
<coap+tcp://simple-host1.example.com/temp>;rt=temperature;ct=0;
    anchor="coap+tcp://simple-host1.example.com"
```

and analogous records.

A.4. A note on differences between link-format and Link headers

While link-format and Link headers look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC5988], which are dealt with differently:

- "Resolving the target against the anchor": [RFC6690] Section 2.1 states that the anchor of a link uses the Base URI against which the term inside the angle brackets (the target) is resolved. [RFC8288] Section B.2 describes that the anchor is immaterial to the resolution of the target reference.

  In the context of a Resource Directory, the authors decided not to not let this become an issue by requiring that RFC6690 links be serialized in a way that either rule set can be applied and give

the same results. Note that all examples of [RFC6690], [RFC8288] and this document comply with that rule.

Applications that would prefer to transport references with a relative target and an absolute anchor are advised to use a different serialization of the links. [I-D.ietf-core-links-json] might provide such formats.

- o There is no percent encoding in link-format documents.

  A link-format document is a UTF-8 encoded string of Unicode characters and does not have percent encoding, while Link headers are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explictly when required.

  For example, while a Link header in a page about a Swedish city might read

  "Link: </temperature/Malm%C3%B6>;rel="live-environment-data""

  a link-format document from the same source might describe the link as

  "</temperature/Malmoe>;rel="live-environment-data""

  Parsers and producers of link-format and header data need to be aware of this difference.

Appendix B. Syntax examples for Protocol Negotiation

[This appendix should not show up in a published version of this document.]

The protocol negotiation that is being worked on in [I-D.silverajan-core-coap-protocol-negotiation] makes use of the Resource Directory.

Until that document is update to use the latest resource-directory specification, here are some examples of protocol negotiation with the current Resource Directory:

An endpoint could register as follows from its address "[2001:db8:f1::2]:5683":

```
   Req: POST coap://rd.example.com/rd?ep=node1
       &at=coap+tcp://[2001:db8:f1::2]
   Content-Format: 40
   Payload:
   </temperature>;ct=0;rt="temperature";if="core.s"

   Res: 2.01 Created
   Location: /rd/1234
```

An endpoint lookup would just reflect the registered attributes:

```
   Req: GET /rd-lookup/ep

   Res: 2.05 Content
   </rd/1234>;ep="node1";con="coap://[2001:db8:f1::2]:5683";
       at="coap+tcp://[2001:db8:f1::2]"
```

A UDP client would then see the following in a resource lookup:

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

Res: 2.05 Content
<coap://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";if="core.s";
    anchor="coap://[2001:db8:f1::2]"
```

while a TCP capable client could say:

```
   Req: GET /rd-lookup/res?rt=temperature&tt=tcp

   Res: 2.05 Content
   <coap+tcp://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";
       if="core.s";anchor="coap+tcp://[2001:db8:f1::2]"
```

Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

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

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Phone: +1-707-502-5136
Email: Michael.Koster@smartthings.com

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

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

Peter van der Stok
consultant

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

Christian Amsuess (editor)
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: christian@amsuess.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 3, 2018

C. Jennings
Cisco
Z. Shelby
ARM
J. Arkko
A. Keranen
Ericsson
C. Bormann
Universitaet Bremen TZI

March 02, 2018

Media Types for Sensor Measurement Lists (SenML) draft-ietf-core-senml-13

Abstract

This specification defines media types for representing simple sensor measurements and device parameters in the Sensor Measurement Lists (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 one of these media types in protocols such as HTTP or CoAP to transport the measurements of the sensor or to be configured.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 3, 2018.

Table of Contents

1. Overview 3
2. Requirements and Design Goals 4
3. Terminology 5
4. SenML Structure and Semantics 6
4.1. Base Fields 6
4.2. Regular Fields 7
4.3. SenML Labels 7
4.4. Considerations 8
4.5. Resolved Records 10
4.6. Associating Meta-data 10
4.7. Configuration and Actuation usage 11
5. JSON Representation (application/senml+json) 11
5.1. Examples 12
5.1.1. Single Datapoint 12
5.1.2. Multiple Datapoints 12
5.1.3. Multiple Measurements 13
5.1.4. Resolved Data 14
5.1.5. Multiple Data Types 15
5.1.6. Collection of Resources 15
5.1.7. Setting an Actuator 16
6. CBOR Representation (application/senml+cbor) 17
7. XML Representation (application/senml+xml) 19
8. EXI Representation (application/senml-exi) 21
9. Fragment Identification Methods 24
9.1. Fragment Identification Examples 24
10. Usage Considerations 25
11. CDDL 26
12. IANA Considerations 27
12.1. Units Registry 27
12.2. SenML Label Registry 31
12.3. Media Type Registrations 32

12.3.1. senml+json Media Type Registration 33
12.3.2. sensml+json Media Type Registration 34
12.3.3. senml+cbor Media Type Registration 35
12.3.4. sensml+cbor Media Type Registration 36
12.3.5. senml+xml Media Type Registration 37
12.3.6. sensml+xml Media Type Registration 38
12.3.7. senml-exi Media Type Registration 40
12.3.8. sensml-exi Media Type Registration 41
12.4. XML Namespace Registration 42
12.5. CoAP Content-Format Registration 42
13. Security Considerations 43
14. Privacy Considerations 43
15. Acknowledgement 43
16. References 44
16.1. Normative References 44
16.2. Informative References 46
Authors' Addresses 48

1. Overview

Connecting sensors to the Internet is not new, and there have been many protocols designed to facilitate it. This specification defines new media types for carrying simple sensor information in a protocol such as HTTP [RFC7230] or CoAP [RFC7252]. 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. SenML can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and the web resource model where the sensor is requested as a resource representation (e.g., "GET /sensor/temperature").

There are many types of more complex measurements and measurements that this media type would not be suitable for. SenML strikes a balance between having some information about the sensor carried with the sensor data so that the data is self describing but it also tries to make that a fairly minimal set of auxiliary information for efficiency reason. Other information about the sensor can be discovered by other methods such as using the CoRE Link Format [RFC6690].

SenML is defined by a data model for measurements and simple meta-data about measurements and devices. The data is structured as a single array that contains a series of SenML Records which can each contain fields such as an unique identifier for the sensor, the time the measurement was made, the unit the measurement is in, and the current value of the sensor. Serializations for this data model are defined for JSON [RFC8259], CBOR [RFC7049], XML

[W3C.REC-xml-20081126], and Efficient XML Interchange (EXI) [W3C.REC-exi-20140211].

For example, the following shows a measurement from a temperature gauge encoded in the JSON syntax.

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","v":23.1}
]
```

In the example above, the array has a single SenML Record with a measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a current value of 23.1 degrees Celsius.

2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets from large numbers of constrained devices. Keeping the total size of payload small makes it easy to use SenML also in constrained networks, e.g., in a 6LoWPAN [RFC4944]. 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 power meters and other large scale deployments has indicated that the solution needs to support allowing multiple measurements to be batched into a single HTTP or CoAP request. This "batch" upload capability allows the server side to efficiently support a large number of devices. It also conveniently supports batch transfers from proxies and storage devices, even in situations where the sensor itself sends just a single data item at a time. The multiple measurements could be from multiple related sensors or from the same sensor but at different times.

The basic design is an array with a series of measurements. The following example shows two measurements made at different times. The value of a measurement is given by the "v" field, the time of a measurement is in the "t" field, the "n" field has a unique sensor name, and the unit of the measurement is carried in the "u" field.

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020076e+09,
   "v":23.5},
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020091e+09,
   "v":23.6}
]
```

To keep the messages small, it does not make sense to repeat the "n" field in each SenML Record so there is a concept of a Base Name which is simply a string that is prepended to the Name field of all

elements in that record and any records that follow it. So a more compact form of the example above is the following.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020076e+09,
   "v":23.5},
  {"u":"Cel","t":1.276020091e+09,
   "v":23.6}
]
```

In the above example the Base Name is in the "bn" field and the "n" fields in each Record are the empty string so they are omitted.

Some devices have accurate time while others do not so SenML supports absolute and relative times. Time is represented in floating point as seconds. Values greater than zero represent an absolute time relative to the Unix epoch (1970-01-01T00:00Z in UTC time) and the time is counted same way as the Portable Operating System Interface (POSIX) "seconds since the epoch" [TIME_T]. Values of 0 or less represent a relative time in the past from the current time. A simple sensor with no absolute wall clock time might take a measurement every second, batch up 60 of them, and then send the batch to a server. It would include the relative time each measurement was made compared to the time the batch was sent in each SenML Record. The server might have accurate NTP time and use the time it received the data, and the relative offset, to replace the times in the SenML with absolute times before saving the SenML Pack in a document database.

3. Terminology

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

This document also uses the following terms:

SenML Record: One measurement or configuration instance in time presented using the SenML data model.

SenML Pack: One or more SenML Records in an array structure.

SenML Label: A short name used in SenML Records to denote different SenML fields (e.g., "v" for "value").

SenML Field: A component of a record that associates a value to a SenML Label for this record.

This document uses the terms "attribute" and "tag" where they occur with the underlying technologies (XML, CBOR [RFC7049], and Link Format [RFC6690]), not for SenML concepts per se. Note that "attribute" has been widely used previously as a synonym for SenML "field", though.

4. SenML Structure and Semantics

Each SenML Pack carries a single array that represents a set of measurements and/or parameters. This array contains a series of SenML Records with several fields described below. There are two kinds of fields: base and regular. The base fields can be included in any SenML Record and they apply to the entries in the Record. Each base field also applies to all Records after it up to, but not including, the next Record that has that same base field. All base fields are optional. Regular fields can be included in any SenML Record and apply only to that Record.

4.1. Base Fields

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

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

Base Unit: A base unit that is assumed for all entries, unless otherwise indicated. If a record does not contain a Unit value, then the Base Unit is used. Otherwise the value found in the Unit (if any) is used.

Base Value: A base value is added to the value found in an entry, similar to Base Time.

Base Sum: A base sum is added to the sum found in an entry, similar to Base Time.

Version: Version number of media type format. This field is an optional positive integer and defaults to 5 if not present. [RFC Editor: change the default value to 10 when this specification is published as an RFC and remove this note]

4.2. Regular Fields

Name: Name of the sensor or parameter. When appended to the Base Name field, this must result in a globally unique identifier for the resource. The name is optional, if the Base Name is present. If the name is missing, Base Name must uniquely identify the resource. This can be used to represent a large array of measurements from the same sensor without having to repeat its identifier on every measurement.

Unit: Unit for a measurement value. Optional.

Value: Value of the entry. Optional if a Sum value is present, otherwise required. Values are represented using basic data types. This specification defines floating point numbers ("v" field for "Value"), booleans ("vb" for "Boolean Value"), strings ("vs" for "String Value") and binary data ("vd" for "Data Value"). Exactly one value field MUST appear unless there is Sum field in which case it is allowed to have no Value field.

Sum: Integrated sum of the values over time. Optional. This field is in the unit specified in the Unit value multiplied by seconds.

Time: Time when value was recorded. Optional.

Update Time: Period of time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. Optional. This can be used to detect the failure of sensors or communications path from the sensor.

4.3. SenML Labels

Table 1 provides an overview of all SenML fields defined by this document with their respective labels and data types.

Name	Label	CBOR Label	JSON Type	XML Type
Base Name	bn	-2	String	string
Base Time	bt	-3	Number	double
Base Unit	bu	-4	String	string
Base Value	bv	-5	Number	double
Base Sum	bs	-6	Number	double
Version	bver	-1	Number	int
Name	n	0	String	string
Unit	u	1	String	string
Value	v	2	Number	double
String Value	vs	3	String	string
Boolean Value	vb	4	Boolean	boolean
Data Value	vd	8	String (*)	string (*)
Value Sum	s	5	Number	double
Time	t	6	Number	double
Update Time	ut	7	Number	double

Table 1: SenML Labels

Data Value is base64 encoded string with URL safe alphabet as defined in Section 5 of [RFC4648], with padding omitted.

For details of the JSON representation see Section 5, for the CBOR Section 6, and for the XML Section 7.

4.4. Considerations

The SenML format can be extended with further custom fields. Both new base and regular fields are allowed. See Section 12.2 for details. Implementations MUST ignore fields they don't recognize unless that field has a label name that ends with the '_' character in which case an error MUST be generated.

All SenML Records in a Pack MUST have the same version number. This is typically done by adding a Base Version field to only the first Record in the Pack.

Systems reading one of the objects MUST check for the Version field. 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 structure or semantics that is different from what is defined in the present document beyond just making use of the extension points provided here. New version numbers can only be defined in an RFC that updates this specification or it successors.

The Name value is concatenated to the Base Name value to yield the name of the sensor. The resulting concatenated name needs to uniquely identify and differentiate the sensor from all others. The concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", "-", ":", ".", "/", and "_"; furthermore, 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 concatenated names can be used directly within various URI schemes (including segments of an HTTP path with no special encoding) and can be used directly in many databases and analytic systems. [RFC5952] contains advice on encoding an IPv6 address in a name. See Section 14 for privacy considerations that apply to the use of long-term stable unique identifiers.

Although it is RECOMMENDED that concatenated names are represented as URIs [RFC3986] or URNs [RFC8141], the restricted character set specified above puts strict limits on the URI schemes and URN namespaces that can be used. As a result, implementers need to take care in choosing the naming scheme for concatenated names, because such names both need to be unique and need to conform to the restricted character set. One approach is to include a bit string that has guaranteed uniqueness (such as a 1-wire address). Some of the examples within this document use the device URN namespace as specified in [I-D.arkko-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name. However, the restricted character set does not allow the use of many URI schemes, such as the 'tag' scheme [RFC4151] and the 'ni' scheme [RFC6920], in names as such. The use of URIs with characters incompatible with this set, and possible mapping rules between the two, are outside of the scope of the present document.

If the Record has no Unit, the Base Unit is used as the Unit. Having no Unit and no Base Unit is allowed.

If either the Base Time or Time value is missing, the missing field 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.

If only one of the Base Sum or Sum value is present, the missing field is considered to have a value of zero. The Base Sum and Sum values are added together to get the sum of measurement. If neither the Base Sum or Sum are present, then the measurement does not have a sum value.

If the Base Value or Value is not present, the missing field(s) are considered to have a value of zero. The Base Value and Value are added together to get the value of the measurement.

Representing the statistical characteristics of measurements, such as accuracy, can be very complex. Future specification may add new fields to provide better information about the statistical properties of the measurement.

In summary, the structure of a SenML record is laid out to support a single measurement per record. If multiple data values are measured at the same time (e.g., air pressure and altitude), they are best kept as separate records linked through their Time value; this is even true where one of the data values is more "meta" than others (e.g., describes a condition that influences other measurements at the same time).

4.5. Resolved Records

Sometimes it is useful to be able to refer to a defined normalized format for SenML records. This normalized format tends to get used for big data applications and intermediate forms when converting to other formats.

A SenML Record is referred to as "resolved" if it does not contain any base values, i.e., labels starting with the character 'b', except for Version fields (see below), and has no relative times. To resolve the records, the base values of the SenML Pack (if any) are applied to the Record. That is, name and base name are concatenated, base time is added to the time of the Record, if the Record did not contain Unit the Base Unit is applied to the record, etc. In addition the records need to be in chronological order. An example of this is show in Section 5.1.4.

The Version field MUST NOT be present in resolved records if the SenML version defined in this document is used and MUST be present otherwise in all the resolved SenML Records.

Future specification that defines new base fields need to specify how the field is resolved.

4.6. Associating Meta-data

SenML is designed to carry the minimum dynamic information about measurements, and for efficiency reasons does not carry significant static meta-data about the device, object or sensors. Instead, it is assumed that this meta-data is carried out of band. For web resources using SenML Packs, this meta-data can be made available

using the CoRE Link Format [RFC6690]. The most obvious use of this link format is to describe that a resource is available in a SenML format in the first place. The relevant media type indicator is included in the Content-Type (ct=) link attribute (which is defined for the Link Format in Section 7.2.1 of [RFC7252]).

4.7. Configuration and Actuation usage

SenML can also be used for configuring parameters and controlling actuators. When a SenML Pack is sent (e.g., using a HTTP/CoAP POST or PUT method) and the semantics of the target are such that SenML is interpreted as configuration/actuation, SenML Records are interpreted as a request to change the values of given (sub)resources (given as names) to given values at the given time(s). The semantics of the target resource supporting this usage can be described, e.g., using [I-D.ietf-core-interfaces]. Examples of actuation usage are shown in Section 5.1.7.

5. JSON Representation (application/senml+json)

For the SenML fields shown in Table 2, the SenML labels are used as the JSON object member names within JSON objects representing the JSON SenML Records.

Name	label	Type
Base Name	bn	String
Base Time	bt	Number
Base Unit	bu	String
Base Value	bv	Number
Base Sum	bs	Number
Version	bver	Number
Name	n	String
Unit	u	String
Value	v	Number
String Value	vs	String
Boolean Value	vb	Boolean
Data Value	vd	String
Value Sum	s	Number
Time	t	Number
Update Time	ut	Number

Table 2: JSON SenML Labels

The root JSON value consists of an array with one JSON object for each SenML Record. All the fields in the above table MAY occur in the records with member values of the type specified in the table.

Only the UTF-8 [RFC3629] form of JSON is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC8259]. Octets in the Data Value are base64 encoded with URL safe alphabet as defined in Section 5 of [RFC4648], with padding omitted.

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]. This allows time values to have better than microsecond precision over the next 100 years. 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. In the interest of avoiding unnecessary verbosity and speeding up processing, the mantissa SHOULD be less than 19 characters long and the exponent SHOULD be less than 5 characters long.

5.1. Examples

5.1.1. Single Datapoint

The following shows a temperature reading taken approximately "now" by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","v":23.1}
]
```

5.1.2. Multiple Datapoints

The following example shows voltage and current now, i.e., at an unspecified time.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:","n":"voltage","u":"V","v":120.1},
  {"n":"current","u":"A","v":1.2}
]
```

The next example is similar to the above one, but shows current at Tue Jun 8 18:01:16.001 UTC 2010 and at each second for the previous 5 seconds.

```
[
  {"bn":"urn:dev:ow:10e2073a0108006:","bt":1.276020076001e+09,
   "bu":"A","bver":5,
   "n":"voltage","u":"V","v":120.1},
  {"n":"current","t":-5,"v":1.2},
  {"n":"current","t":-4,"v":1.3},
  {"n":"current","t":-3,"v":1.4},
  {"n":"current","t":-2,"v":1.5},
  {"n":"current","t":-1,"v":1.6},
  {"n":"current","v":1.7}
]
```

Note that in some usage scenarios of SenML the implementations MAY store or transmit SenML in a stream-like fashion, where data is collected over time and continuously added to the object. This mode of operation is optional, but systems or protocols using SenML in this fashion MUST specify that they are doing this. SenML defines separate media types to indicate Sensor Streaming Measurement Lists (SensML) for this usage (see Section 12.3.2). In this situation the SensML stream can be sent and received in a partial fashion, i.e., a measurement entry can be read as soon as the SenML Record is received and not have to wait for the full SensML Stream to be complete.

For instance, the following stream of measurements may be sent via a long lived HTTP POST from the producer of a SensML to the consumer of that, and each measurement object may be reported at the time it was measured:

```
[
  {"bn":"urn:dev:ow:10e2073a01080063","bt":1.320067464e+09,
   "bu":"%RH","v":21.2},
  {"t":10,"v":21.3},
  {"t":20,"v":21.4},
  {"t":30,"v":21.4},
  {"t":40,"v":21.5},
  {"t":50,"v":21.5},
  {"t":60,"v":21.5},
  {"t":70,"v":21.6},
  {"t":80,"v":21.7},
...
```

5.1.3. Multiple Measurements

The following example shows humidity measurements from a mobile device with a 1-wire address 10e2073a01080063, starting at Mon Oct 31 13:24:24 UTC 2011. The device also provides position data, which is provided in the same measurement or parameter array as separate entries. Note time is used to for correlating data that belongs

together, e.g., a measurement and a parameter associated with it. Finally, the device also reports extra data about its battery status at a separate time.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063","bt":1.320067464e+09,
   "bu":"%RH","v":20},
  {"u":"lon","v":24.30621},
  {"u":"lat","v":60.07965},
  {"t":60,"v":20.3},
  {"u":"lon","t":60,"v":24.30622},
  {"u":"lat","t":60,"v":60.07965},
  {"t":120,"v":20.7},
  {"u":"lon","t":120,"v":24.30623},
  {"u":"lat","t":120,"v":60.07966},
  {"u":"%EL","t":150,"v":98},
  {"t":180,"v":21.2},
  {"u":"lon","t":180,"v":24.30628},
  {"u":"lat","t":180,"v":60.07967}
]
```

The size of this example represented in various forms, as well as that form compressed with gzip is given in the following table.

Encoding	Size	Compressed Size
JSON	573	206
XML	649	235
CBOR	254	196
EXI	161	184

Table 3: Size Comparisons

5.1.4. Resolved Data

The following shows the example from the previous section show in resolved format.

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067464e+09,
   "v":20},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067464e+09,
   "v":24.30621},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067464e+09,
   "v":60.07965},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067524e+09,
   "v":20.3},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067524e+09,
   "v":24.30622},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067524e+09,
   "v":60.07965},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067584e+09,
   "v":20.7},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067584e+09,
   "v":24.30623},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067584e+09,
   "v":60.07966},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%EL","t":1.320067614e+09,
   "v":98},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067644e+09,
   "v":21.2},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067644e+09,
   "v":24.30628},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067644e+09,
   "v":60.07967}
]
```

5.1.5. Multiple Data Types

The following example shows a sensor that returns different data types.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:","n":"temp","u":"Cel","v":23.1},
  {"n":"label","vs":"Machine Room"},
  {"n":"open","vb":false},
  {"n":"nfv-reader","vd":"aGkgCg"}
]
```

5.1.6. Collection of Resources

The following example shows the results from a query to one device that aggregates multiple measurements from another devices. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on http://[2001:db8::2] at Mon Oct 31 16:27:09 UTC 2011, and has gotten two separate values

as a result, a temperature and humidity measurement as well as the results from another device at http://[2001:db8::1] that also had a temperature and humidity. Note that the last record would use the Base Name from the 3rd record but the Base Time from the first record.

```
[
  {"bn":"2001:db8::2/","bt":1.320078429e+09,
   "n":"temperature","u":"Cel","v":25.2},
  {"n":"humidity","u":"%RH","v":30},
  {"bn":"2001:db8::1/","n":"temperature","u":"Cel","v":12.3},
  {"n":"humidity","u":"%RH","v":67}
]
```

5.1.7. Setting an Actuator

The following example show the SenML that could be used to set the current set point of a typical residential thermostat which has a temperature set point, a switch to turn on and off the heat, and a switch to turn on the fan override.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:"},
  {"n":"temp","u":"Cel","v":23.1},
  {"n":"heat","u":"/","v":1},
  {"n":"fan","u":"/","v":0}
]
```

In the following example two different lights are turned on. It is assumed that the lights are on a network that can guarantee delivery of the messages to the two lights within 15 ms (e.g. a network using 802.1BA [IEEE802.1ba-2011] and 802.1AS [IEEE802.1as-2011] for time synchronization). The controller has set the time of the lights coming on to 20 ms in the future from the current time. This allows both lights to receive the message, wait till that time, then apply the switch command so that both lights come on at the same time.

```
[
  {"bt":1.320078429e+09,"bu":"/","n":"2001:db8::3","v":1},
  {"n":"2001:db8::4","v":1}
]
```

The following shows two lights being turned off using a non deterministic network that has a high odds of delivering a message in less than 100 ms and uses NTP for time synchronization. The current time is 1320078429. The user has just turned off a light switch which is turning off two lights. Both lights are dimmed to 50% brightness immediately to give the user instant feedback that

something is changing. However given the network, the lights will probably dim at somewhat different times. Then 100 ms in the future, both lights will go off at the same time. The instant but not synchronized dimming gives the user the sensation of quick responses and the timed off 100 ms in the future gives the perception of both lights going off at the same time.

```
[
  {"bt":1.320078429e+09,"bu":"/","n":"2001:db8::3","v":0.5},
  {"n":"2001:db8::4","v":0.5},
  {"n":"2001:db8::3","t":0.1,"v":0},
  {"n":"2001:db8::4","t":0.1,"v":0}
]
```

6. CBOR Representation (application/senml+cbor)

The CBOR [RFC7049] representation is equivalent to the JSON representation, with the following changes:

- For JSON Numbers, the CBOR representation can use integers, floating point numbers, or decimal fractions (CBOR Tag 4); however a representation SHOULD be chosen such that when the CBOR value is converted back to an IEEE double precision floating point value, it has exactly the same value as the original Number. For the version number, only an unsigned integer is allowed.

- Characters in the String Value are encoded using a definite length text string (type 3). Octets in the Data Value are encoded using a definite length byte string (type 2).

- For compactness, the CBOR representation uses integers for the labels, as defined in Table 4. This table is conclusive, i.e., there is no intention to define any additional integer map keys; any extensions will use string map keys. This allows translators converting between CBOR and JSON representations to convert also all future labels without needing to update implementations.

Name	Label	CBOR Label
Version	bver	-1
Base Name	bn	-2
Base Time	bt	-3
Base Unit	bu	-4
Base Value	bv	-5
Base Sum	bs	-6
Name	n	0
Unit	u	1
Value	v	2
String Value	vs	3
Boolean Value	vb	4
Value Sum	s	5
Time	t	6
Update Time	ut	7
Data Value	vd	8

Table 4: CBOR representation: integers for map keys

- For streaming SensML in CBOR representation, the array containing the records SHOULD be a CBOR indefinite length array while for non-streaming SenML, a definite length array MUST be used.

The following example shows a dump of the CBOR example for the same sensor measurement as in Section 5.1.2.

```
0000 87 a7 21 78 1b 75 72 6e 3a 64 65 76 3a 6f 77 3a |..!x.urn:dev:ow:|
0010 31 30 65 32 30 37 33 61 30 31 30 38 30 30 36 3a |10e2073a0108006:|
0020 22 fb 41 d3 03 a1 5b 00 10 62 23 61 41 20 05 00 |".A...[..b#aA ..|
0030 67 76 6f 6c 74 61 67 65 01 61 56 02 fb 40 5e 06 |gvoltage.aV..@^.|
0040 66 66 66 66 66 a3 00 67 63 75 72 72 65 6e 74 06 |fffff..gcurrent.|
0050 24 02 fb 3f f3 33 33 33 33 33 33 a3 00 67 63 75 |$..?.333333..gcu|
0060 72 72 65 6e 74 06 23 02 fb 3f f4 cc cc cc cc cc |rrent.#..?......|
0070 cd a3 00 67 63 75 72 72 65 6e 74 06 22 02 fb 3f |...gcurrent."..?|
0080 f6 66 66 66 66 66 66 a3 00 67 63 75 72 72 65 6e |.ffffff..gcurren|
0090 74 06 21 02 f9 3e 00 a3 00 67 63 75 72 72 65 6e |t.!..>...gcurren|
00a0 74 06 20 02 fb 3f f9 99 99 99 99 99 9a a3 00 67 |t. ..?.........g|
00b0 63 75 72 72 65 6e 74 06 00 02 fb 3f fb 33 33 33 |current....?.333|
00c0 33 33 33                                       |333|
00c3
```

In CBOR diagnostic notation (Section 6 of [RFC7049]), this is:

```
[{-2: "urn:dev:ow:10e2073a0108006:",
  -3: 1276020076.001, -4: "A", -1: 5, 0: "voltage", 1: "V", 2: 120.1},
 {0: "current", 6: -5, 2: 1.2}, {0: "current", 6: -4, 2: 1.3},
 {0: "current", 6: -3, 2: 1.4}, {0: "current", 6: -2, 2: 1.5},
 {0: "current", 6: -1, 2: 1.6}, {0: "current", 6: 0, 2: 1.7}]
```

7. XML Representation (application/senml+xml)

A SenML Pack or Stream can also be represented in XML format as defined in this section.

Only the UTF-8 form of XML is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC8259]. Octets in the Data Value are base64 encoded with URL safe alphabet as defined in Section 5 of [RFC4648].

The following example shows an XML example for the same sensor measurement as in Section 5.1.2.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a0108006:" bt="1.276020076001e+09"
  bu="A" bver="5" n="voltage" u="V" v="120.1"></senml>
  <senml n="current" t="-5" v="1.2"></senml>
  <senml n="current" t="-4" v="1.3"></senml>
  <senml n="current" t="-3" v="1.4"></senml>
  <senml n="current" t="-2" v="1.5"></senml>
  <senml n="current" t="-1" v="1.6"></senml>
  <senml n="current" v="1.7"></senml>
</sensml>
```

The SenML Stream is represented as a sensml element that contains a series of senml elements for each SenML Record. The SenML fields are represented as XML attributes. For each field defined in this document, the following table shows the SenML labels, which are used for the XML attribute name, as well as the according restrictions on the XML attribute values ("type") as used in the XML senml elements.

Name	Label	Type
Base Name	bn	string
Base Time	bt	double
Base Unit	bu	string
Base Value	bv	double
Base Sum	bs	double
Base Version	bver	int
Name	n	string
Unit	u	string
Value	v	double
String Value	vs	string
Data Value	vd	string
Boolean Value	vb	boolean
Value Sum	s	double
Time	t	double
Update Time	ut	double

Table 5: XML SenML Labels

The RelaxNG [RNC] schema for the XML is:

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

senml = element senml {
  attribute bn { xsd:string }?,
  attribute bt { xsd:double }?,
  attribute bv { xsd:double }?,
  attribute bs { xsd:double }?,
  attribute bu { xsd:string }?,
  attribute bver { xsd:int }?,

  attribute n { xsd:string }?,
  attribute s { xsd:double }?,
  attribute t { xsd:double }?,
  attribute u { xsd:string }?,
  attribute ut { xsd:double }?,

  attribute v { xsd:double }?,
  attribute vb { xsd:boolean }?,
  attribute vs { xsd:string }?,
  attribute vd { xsd:string }?
}

sensml =
  element sensml {
    senml+
}

start = sensml
```

8. EXI Representation (application/senml-exi)

For efficient transmission of SenML over e.g. a constrained network, Efficient XML Interchange (EXI) can be used. This encodes the XML Schema [W3C.REC-xmlschema-1-20041028] 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 MUST include an "EXI Options", as defined in [W3C.REC-exi-20140211], with an schemaId set to the value of "a" indicating the schema provided in this specification. Future revisions to the schema can change the value of the 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 is the XSD Schema to be used for strict schema guided EXI processing. It is generated from the RelaxNG.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="urn:ietf:params:xml:ns:senml"
xmlns:ns1="urn:ietf:params:xml:ns:senml">
  <xs:element name="senml">
    <xs:complexType>
      <xs:attribute name="bn" type="xs:string" />
      <xs:attribute name="bt" type="xs:double" />
      <xs:attribute name="bv" type="xs:double" />
      <xs:attribute name="bs" type="xs:double" />
      <xs:attribute name="bu" type="xs:string" />
      <xs:attribute name="bver" type="xs:int" />
      <xs:attribute name="n" type="xs:string" />
      <xs:attribute name="s" type="xs:double" />
      <xs:attribute name="t" type="xs:double" />
      <xs:attribute name="u" type="xs:string" />
      <xs:attribute name="ut" type="xs:double" />
      <xs:attribute name="v" type="xs:double" />
      <xs:attribute name="vb" type="xs:boolean" />
      <xs:attribute name="vs" type="xs:string" />
      <xs:attribute name="vd" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="sensml">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="ns1:senml" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The following shows a hexdump of the EXI produced from encoding the following XML example. Note this example is the same information as the first example in Section 5.1.2 in JSON format.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a01080063:" n="voltage" u="V"
  v="120.1"></senml>
  <senml n="current" u="A" v="1.2"></senml>
</sensml>
```

Which compresses with EXI to the following displayed in hexdump:

```
0000 a0 30 0d 84 80 f3 ab 93 71 d3 23 2b b1 d3 7b b9 |.0......q.#+..{.|
0010 d1 89 83 29 91 81 b9 9b 09 81 89 81 c1 81 81 b1 |...)............|
0020 99 d2 84 bb 37 b6 3a 30 b3 b2 90 1a b1 58 84 c0 |....7.:0.....X..|
0030 33 04 b1 ba b9 39 32 b7 3a 10 1a 09 06 40 38    |3....92.:....@8|
003f
```

The above example used the bit packed form of EXI but it is also possible to use a byte packed form of EXI which can makes it easier for a simple sensor to produce valid EXI without really implementing EXI. Consider the example of a temperature sensor that produces a value in tenths of degrees Celsius over a range of 0.0 to 55.0. It would produce an XML SenML file such as:

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml n="urn:dev:ow:10e2073a01080063" u="Cel" v="23.1"></senml>
</sensml>
```

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

```
0000 a0 00 48 80 6c 20 01 06 1d 75 72 6e 3a 64 65 76 |..H.l ...urn:dev|
0010 3a 6f 77 3a 31 30 65 32 30 37 33 61 30 31 30 38 |:ow:10e2073a0108|
0020 30 30 36 33 02 05 43 65 6c 01 00 e7 01 01 00 03 |0063..Cel.......|
0030 01                                              |.|
0031
```

A small temperature sensor device that only generates this one EXI file does not really need an full EXI implementation. It can simply hard code the output replacing the 1-wire device ID starting at byte 0x20 and going to byte 0x2F with it's device ID, and replacing the value "0xe7 0x01" at location 0x37 and 0x38 with the current temperature. The EXI Specification [W3C.REC-exi-20140211] contains the full information on how floating point numbers are represented, but for the purpose of this sensor, the temperature can be converted to an integer in tenths of degrees (231 in this example). EXI stores 7 bits of the integer in each byte with the top bit set to one if there are further bytes. So the first bytes at is set to low 7 bits of the integer temperature in tenths of degrees plus 0x80. In this example 231 & 0x7F + 0x80 = 0xE7. The second byte is set to the integer temperature in tenths of degrees right shifted 7 bits. In this example 231 >> 7 = 0x01.

9. Fragment Identification Methods

A SenML Pack typically consists of multiple SenML Records and for some applications it may be useful to be able to refer with a Fragment Identifier to a single record, or a set of records, in a Pack. The fragment identifier is only interpreted by a client and does not impact retrieval of a representation. The SenML Fragment Identification is modeled after CSV Fragment Identifiers [RFC7111].

To select a single SenML Record, the "rec" scheme followed by a single number is used. For the purpose of numbering records, the first record is at position 1. A range of records can be selected by giving the first and the last record number separated by a '-' character. Instead of the second number, the '*' character can be used to indicate the last SenML Record in the Pack. A set of records can also be selected using a comma separated list of record positions or ranges.

(We use the term "selecting a record" for identifying it as part of the fragment, not in the sense of isolating it from the Pack -- the record still needs to be interpreted as part of the Pack, e.g., using the base values defined in earlier records)

9.1. Fragment Identification Examples

The 3rd SenML Record from "coap://example.com/temp" resource can be selected with:

coap://example.com/temp#rec=3

Records from 3rd to 6th can be selected with:

coap://example.com/temp#rec=3-6

Records from 19th to the last can be selected with:

coap://example.com/temp#rec=19-*

The 3rd and 5th record can be selected with:

coap://example.com/temp#rec=3,5

To select the Records from third to fifth, the 10th record, and all from 19th to the last:

coap://example.com/temp#rec=3-5,10,19-*

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 unit set to watts, but it would put the sum of energy used in the "s" field of the measurement. It might optionally include the current power in the "v" field.

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. Implementers are encouraged to report the cumulative sum as well as the raw value of a given sensor.

Applications that use the cumulative sum values need to understand they are very loosely defined by this specification, and depending on the particular sensor implementation may behave in unexpected ways. Applications should be able to deal with the following issues:

1. Many sensors will allow the cumulative sums to "wrap" back to zero after the value gets sufficiently large.

2. Some sensors will reset the cumulative sum back to zero when the device is reset, loses power, or is replaced with a different sensor.

3. Applications cannot make assumptions about when the device started accumulating values into the sum.

Typically applications can make some assumptions about specific sensors that will allow them to deal with these problems. A common assumption is that for sensors whose measurement values are always positive, the sum should never get smaller; so if the sum does get smaller, the application will know that one of the situations listed above has happened.

11. CDDL

As a convenient reference, the JSON and CBOR representations can be described with the common CDDL [I-D.ietf-cbor-cddl] specification in Figure 1 (informative).

```
SenML-Pack = [1* record]

record = {
  ? bn => tstr,         ; Base Name
  ? bt => numeric,      ; Base Time
  ? bu => tstr,         ; Base Units
  ? bv => numeric,      ; Base Value
  ? bs => numeric,      ; Base Sum
  ? bver => uint,       ; Base Version
  ? n => tstr,         ; Name
  ? u => tstr,         ; Units
  ? s => numeric,      ; Value Sum
  ? t => numeric,      ; Time
  ? ut => numeric,     ; Update Time
  ? ( v => numeric // ; Numeric Value
      vs => tstr //   ; String Value
      vb => bool //   ; Boolean Value
      vd => binary-value ) ; Data Value
  * key-value-pair
}

; now define the generic versions
key-value-pair = ( label => value )

label = non-b-label / b-label
non-b-label = tstr .regexp  "[A-Zac-z0-9][-_:.A-Za-z0-9]*" / uint
b-label = tstr .regexp  "b[-_:.A-Za-z0-9]+" / nint

value = tstr / binary-value / numeric / bool
numeric = number / decfrac
```

Figure 1: Common CDDL specification for CBOR and JSON SenML

For JSON, we use text labels and base64url-encoded binary data (Figure 2).

```
bver = "bver" n  = "n"   s  = "s"
bn  = "bn"    u  = "u"   t  = "t"
bt  = "bt"    v  = "v"   ut = "ut"
bu  = "bu"    vs = "vs"  vd = "vd"
bv  = "bv"    vb = "vb"
bs  = "bs"

binary-value = tstr             ; base64url encoded
```

Figure 2: JSON-specific CDDL specification for SenML

For CBOR, we use integer labels and native binary data (Figure 3).

```
bver = -1  n  = 0   s  = 5
bn  = -2   u  = 1   t  = 6
bt  = -3   v  = 2   ut = 7
bu  = -4   vs = 3   vd = 8
bv  = -5   vb = 4
bs  = -6

binary-value = bstr
```

Figure 3: CBOR-specific CDDL specification for SenML

12. IANA Considerations

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

12.1. Units Registry

IANA will create a registry of SenML unit symbols. The primary purpose of this registry is to make sure that symbols uniquely map to give type of measurement. Definitions for many of these units can be found in location such as [NIST811] and [BIPM]. Units marked with an asterisk are NOT RECOMMENDED to be produced by new implementations, but are in active use and SHOULD be implemented by consumers that can use the related base units.

Symbol	Description	Type	Reference
m	meter	float	RFC-AAAA
kg	kilogram	float	RFC-AAAA
g	gram*	float	RFC-AAAA
s	second	float	RFC-AAAA
A	ampere	float	RFC-AAAA
K	kelvin	float	RFC-AAAA

cd	candela	float	RFC-AAAA
mol	mole	float	RFC-AAAA
Hz	hertz	float	RFC-AAAA
rad	radian	float	RFC-AAAA
sr	steradian	float	RFC-AAAA
N	newton	float	RFC-AAAA
Pa	pascal	float	RFC-AAAA
J	joule	float	RFC-AAAA
W	watt	float	RFC-AAAA
C	coulomb	float	RFC-AAAA
V	volt	float	RFC-AAAA
F	farad	float	RFC-AAAA
Ohm	ohm	float	RFC-AAAA
S	siemens	float	RFC-AAAA
Wb	weber	float	RFC-AAAA
T	tesla	float	RFC-AAAA
H	henry	float	RFC-AAAA
Cel	degrees Celsius	float	RFC-AAAA
lm	lumen	float	RFC-AAAA
lx	lux	float	RFC-AAAA
Bq	becquerel	float	RFC-AAAA
Gy	gray	float	RFC-AAAA
Sv	sievert	float	RFC-AAAA
kat	katal	float	RFC-AAAA
m2	square meter (area)	float	RFC-AAAA
m3	cubic meter (volume)	float	RFC-AAAA
l	liter (volume)*	float	RFC-AAAA
m/s	meter per second (velocity)	float	RFC-AAAA
m/s2	meter per square second (acceleration)	float	RFC-AAAA
m3/s	cubic meter per second (flow rate)	float	RFC-AAAA
l/s	liter per second (flow rate)*	float	RFC-AAAA
W/m2	watt per square meter (irradiance)	float	RFC-AAAA
cd/m2	candela per square meter (luminance)	float	RFC-AAAA
bit	bit (information content)	float	RFC-AAAA
bit/s	bit per second (data rate)	float	RFC-AAAA
lat	degrees latitude (note 1)	float	RFC-AAAA
lon	degrees longitude (note 1)	float	RFC-AAAA
pH	pH value (acidity; logarithmic quantity)	float	RFC-AAAA
dB	decibel (logarithmic quantity)	float	RFC-AAAA
dBW	decibel relative to 1 W (power level)	float	RFC-AAAA
Bspl	bel (sound pressure level; logarithmic quantity)*	float	RFC-AAAA
count	1 (counter value)	float	RFC-AAAA
/	1 (Ratio e.g., value of a switch,	float	RFC-AAAA

	note 2)		
%	1 (Ratio e.g., value of a switch, note 2)*	float	RFC-AAAA
%RH	Percentage (Relative Humidity)	float	RFC-AAAA
%EL	Percentage (remaining battery energy level)	float	RFC-AAAA
EL	seconds (remaining battery energy level)	float	RFC-AAAA
1/s	1 per second (event rate)	float	RFC-AAAA
1/min	1 per minute (event rate, "rpm")*	float	RFC-AAAA
beat/min	1 per minute (Heart rate in beats per minute)*	float	RFC-AAAA
beats	1 (Cumulative number of heart beats)*	float	RFC-AAAA
S/m	Siemens per meter (conductivity)	float	RFC-AAAA

Table 6

- Note 1: Assumed to be in WGS84 unless another reference frame is known for the sensor.

- Note 2: A value of 0.0 indicates the switch is off while 1.0 indicates on and 0.5 would be half on. The preferred name of this unit is "/". For historical reasons, the name "%" is also provided for the same unit - but note that while that name strongly suggests a percentage (0..100) -- it is however NOT a percentage, but the absolute ratio!

New entries can be added to the registration by Expert Review as defined in [RFC8126]. 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. Each unit should define the semantic information and be chosen carefully. Implementers 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 recommended. Instead one can represent the value using scientific notation such a 1.2e3. The "kg" unit is exception to this rule since it is an SI base unit; the "g" unit is provided for legacy compatibility.

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.

   (Note that some amount of judgment will be required here, as even SI itself is not entirely consistent in this respect. For instance, for temperature [ISO-80000-5] defines a quantity, item 5-1 (thermodynamic temperature), and a corresponding unit 5-1.a (Kelvin), and then goes ahead to define another quantity right besides that, item 5-2 ("Celsius temperature"), and the corresponding unit 5-2.a (degree Celsius). The latter quantity is defined such that it gives the thermodynamic temperature as a delta from T0 = 275.15 K. ISO 80000-5 is defining both units side by side, and not really expressing a preference. This level of recognition of the alternative unit degree Celsius is the reason why Celsius temperatures exceptionally seem acceptable in the SenML units list alongside Kelvin.)

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, mph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, kWh.

9. The unit names are case sensitive and the correct case needs to be used, but symbols that differ only in case should not be allocated.

10. A number after a unit typically indicates the previous unit raised to that power, and the / indicates that the units that follow are the reciprocal. A unit should have only one / in the name.

11. A good list of common units can be found in the Unified Code for Units of Measure [UCUM].

12.2. SenML Label Registry

IANA will create a new registry for SenML labels. The initial content of the registry is:

Name	Label	CL	JSON Type	XML Type	EI	Reference
Base Name	bn	-2	String	string	a	RFCXXXX
Base Time	bt	-3	Number	double	a	RFCXXXX
Base Unit	bu	-4	String	string	a	RFCXXXX
Base Value	bv	-5	Number	double	a	RFCXXXX
Base Sum	bs	-6	Number	double	a	RFCXXXX
Base Version	bver	-1	Number	int	a	RFCXXXX
Name	n	0	String	string	a	RFCXXXX
Unit	u	1	String	string	a	RFCXXXX
Value	v	2	Number	double	a	RFCXXXX
String Value	vs	3	String	string	a	RFCXXXX
Boolean Value	vb	4	Boolean	boolean	a	RFCXXXX
Data Value	vd	8	String	string	a	RFCXXXX
Value Sum	s	5	Number	double	a	RFCXXXX
Time	t	6	Number	double	a	RFCXXXX
Update Time	ut	7	Number	double	a	RFCXXXX

Table 7: IANA Registry for SenML Labels, CL = CBOR Label, EI = EXI ID

This is the same table as Table 1, with notes removed, and with columns added for the information that is all the same for this initial set of registrations, but will need to be supplied with a different value for new registrations.

Note to RFC Editor. Please replace RFCXXXX with the number for this RFC.

All new entries must define the Label Name, Label, and XML Type but the CBOR labels SHOULD be left empty as CBOR will use the string encoding for any new labels. The EI column contains the EXI schemaId value of the first Schema which includes this label or is empty if this label was not intended for use with EXI. The Note field SHOULD contain information about where to find out more information about this label.

The JSON, CBOR, and EXI types are derived from the XML type. All XML numeric types such as double, float, integer and int become a JSON Number. XML boolean and string become a JSON Boolean and String respectively. CBOR represents numeric values with a CBOR type that does not lose any information from the JSON value. EXI uses the XML types.

New entries can be added to the registration by Expert Review as defined in [RFC8126]. Experts should exercise their own good judgment but need to consider that shorter labels should have more strict review. New entries should not be made that counteract the advice at the end of Section 4.4.

All new SenML labels that have "base" semantics (see Section 4.1) MUST start with the character 'b'. Regular labels MUST NOT start with that character.

Extensions that add a label that is intended for use with XML need to create a new RelaxNG scheme that includes all the labels in the IANA registry.

Extensions that add a label that is intended for use with EXI need to create a new XSD Schema that includes all the labels in the IANA registry and then allocate a new EXI schemaId value. Moving to the next letter in the alphabet is the suggested way to create the new value for the EXI schemaId. Any labels with previously blank ID values SHOULD be updated in the IANA table to have their ID set to this new schemaId value.

Extensions that are mandatory to understand to correctly process the Pack MUST have a label name that ends with the '_' character.

12.3. Media Type Registrations

The following registrations are done following the procedure specified in [RFC6838] and [RFC7303]. This document registers media types for each serialization format of SenML (JSON, CBOR, and EXI) and also media types for the same formats of the streaming use (SensML). Clipboard formats are defined for the JSON and XML form of lists but do not make sense for streams or other formats.

Note to RFC Editor - please remove this paragraph. Note that a request for media type review for senml+json was sent to the media-types@iana.org on Sept 21, 2010. A second request for all the types was sent on October 31, 2016. Please change all instances of RFC-AAAA with the RFC number of this document.

12.3.1. 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 [RFC8259]. See RFC-AAAA for details. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any JSON key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" 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 e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+json is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senml

Windows Clipboard Name: "JSON Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-json conforms to public.text

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.2. sensml+json Media Type Registration

Type name: application

Subtype name: sensml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. See RFC-AAAA for details. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any JSON key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" 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 e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+json is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.3. senml+cbor Media Type Registration

Type name: application

Subtype name: senml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+cbor is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmlc

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-cbor conforms to public.data

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.4. sensml+cbor Media Type Registration

Type name: application

Subtype name: sensml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+cbor is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmlc

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.5. senml+xml Media Type Registration

Type name: application

Subtype name: senml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be

generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+xml is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmlx

Windows Clipboard Name: "XML Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-xml conforms to public.xml

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.6. sensml+xml Media Type Registration

Type name: application

Subtype name: sensml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+xml is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmlx

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.7. senml-exi Media Type Registration

Type name: application

Subtype name: senml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack. Further information on using schemas to guide the EXI can be found in RFC-AAAA.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml-exi is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmle

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-exi conforms to public.data

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.8. sensml-exi Media Type Registration

Type name: application

Subtype name: sensml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack. Further information on using schemas to guide the EXI can be found in RFC-AAAA.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml-exi is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmle

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.4. XML Namespace Registration

This document registers the following XML namespaces in the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:senml

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces

12.5. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the SenML media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. All IDs are assigned from the "Expert Review" (0-255) range. The assigned IDs are show in Table 8.

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

Table 8: CoAP Content-Format IDs

13. Security Considerations

Sensor data can contain a wide range of information ranging from information that is very public, such as the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. The SenML formats do not provide any security and instead rely on the protocol that carries them to provide security. Applications using SenML need to look at the overall context of how these media types will be used to decide if the security is adequate. The SenML formats defined by this specification do not contain any executable content. However, future extensions could potentially embed application specific executable content in the data.

See also Section 14.

14. Privacy Considerations

Sensor data can range from information with almost no security considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transport protocol such as S/MIME [RFC5751] or HTTP with TLS [RFC5246] that can provide integrity, confidentiality, and authentication information about the source of the data.

The name fields need to uniquely identify the sources or destinations of the values in a SenML Pack. However, the use of long-term stable unique identifiers can be problematic for privacy reasons [RFC6973], depending on the application and the potential of these identifiers to be used in correlation with other information. They should be used with care or avoided as for example described for IPv6 addresses in [RFC7721].

15. Acknowledgement

We would like to thank Alexander Pelov, Alexey Melnikov, Andrew McClure, Andrew McGregor, Bjoern Hoehrmann, Christian Amsuess, Christian Groves, Daniel Peintner, Jan-Piet Mens, Jim Schaad, Joe Hildebrand, John Klensin, Karl Palsson, Lennart Duhrsen, Lisa Dusseault, Lyndsay Campbell, Martin Thomson, Michael Koster, Peter Saint-Andre, and Stephen Farrell, for their review comments.

16. References

16.1. Normative References

[BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI)", 8th edition, 2006.

[IEEE.754.1985] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE Standard 754, August 1985.

[NIST811] Thompson, A. and B. Taylor, "Guide for the Use of the International System of Units (SI)", NIST Special Publication 811, 2008.

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

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

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

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <https://www.rfc-editor.org/info/rfc7303>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

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

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

[RNC] ISO/IEC, "Information technology -- Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG", ISO/IEC 19757-2, Annex C: RELAX NG Compact syntax, December 2008.

[TIME_T] The Open Group Base Specifications, "Vol. 1: Base Definitions, Issue 7", Section 4.15 'Seconds Since the Epoch', IEEE Std 1003.1, 2013 Edition, 2013, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_15>.

[W3C.REC-exi-20140211] Schneider, J., Kamiya, T., Peintner, D., and R. Kyusakov, "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)", World Wide Web Consortium Recommendation REC-exi-20140211, February 2014, <http://www.w3.org/TR/2014/REC-exi-20140211>.

[W3C.REC-xml-20081126] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

[W3C.REC-xmlschema-1-20041028]
Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

16.2. Informative References

[I-D.arkko-core-dev-urn]
Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-05 (work in progress), October 2017.

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-10 (work in progress), September 2017.

[IEEE802.1as-2011]
IEEE, "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks", 2011.

[IEEE802.1ba-2011]
IEEE, "IEEE Standard for Local and metropolitan area networks--Audio Video Bridging (AVB) Systems", 2011.

[ISO-80000-5]
"Quantities and units - Part 5: Thermodynamics", ISO 80000-5, Edition 1.0, May 2007.

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

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", RFC 4151, DOI 10.17487/RFC4151, October 2005, <https://www.rfc-editor.org/info/rfc4151>.

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

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

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <https://www.rfc-editor.org/info/rfc5751>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

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

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[RFC7111] Hausenblas, M., Wilde, E., and J. Tennison, "URI Fragment Identifiers for the text/csv Media Type", RFC 7111, DOI 10.17487/RFC7111, January 2014, <https://www.rfc-editor.org/info/rfc7111>.

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

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics, 2013, <http://unitsofmeasure.org/ucum.html>.

Authors' Addresses

Cullen Jennings
Cisco
400 3rd Avenue SW
Calgary, AB T2P 4H2
Canada

Email: fluffy@iii.ca

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

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

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

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

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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: June 4, 2018

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio

December 01, 2017

YANG Schema Item iDentifier (SID)
draft-ietf-core-sid-03

Abstract

YANG Schema Item iDentifiers (SID) are globally unique 64-bit unsigned numbers used to identify YANG items. This document defines the semantics, the registration, and assignment processes of SIDs. To enable the implementation of these processes, this document also defines a file format used to persist and publish assigned SIDs.

Status of This Memo

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

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

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

This Internet-Draft will expire on June 4, 2018.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology and Notation 3
3. ".sid" file lifecycle 4
4. ".sid" file format 7
5. Security Considerations 11
6. IANA Considerations 11
6.1. "SID mega-range" registry 11
6.1.1. IANA SID Mega-Range Registry 12
6.1.2. IANA "RFC SID range assignment" sub-registries . . . 13
6.2. "YANG module assignment" registry 14
7. Acknowledgments 14
8. References 14
8.1. Normative References 14
8.2. Informative References 15
Appendix A. ".sid" file example 16
Authors' Addresses 25

1. Introduction

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called SID, is encoded using a 64-bit unsigned integer. The following items are identified using SIDs:

- identities
- data nodes
- RPCs and associated input(s) and output(s)
- actions and associated input(s) and output(s)
- notifications and associated information
- YANG modules, submodules and features

To minimize their size, SIDs are often represented as a difference between the current SID and a reference SID. Such difference is called "delta", shorthand for "delta-encoded SID". Conversion from

SIDs to deltas and back to SIDs is a stateless process. Each protocol implementing deltas must unambiguously define the reference SID for each YANG item.

SIDs are globally unique numbers, a registration system is used in order to guarantee their uniqueness. SIDs are registered in blocks called "SID ranges".

Assignment of SIDs to YANG items can be automated, the recommended process to assign SIDs is as follows:

1. A tool extracts the different items defined for a specific YANG module.

2. The list of items is sorted in alphabetical order, 'namespace' in descending order, 'identifier' in ascending order. The 'namespace' and 'identifier' formats are described in the YANG module 'ietf-sid-file' defined in Section 4.

3. SIDs are assigned sequentially from the entry point up to the size of the registered SID range. This approach is recommended to minimize the serialization overhead, especially when delta encoding is implemented.

4. If the number of items exceeds the SID range(s) allocated to a YANG module, an extra range is added for subsequent assignments.

SIDs are assigned permanently, items introduced by a new revision of a YANG module are added to the list of SIDs already assigned. This process can also be automated using the same method described above, only unassigned YANG items are processed at step #3.

Section 3 provides more details about the registration process of YANG modules and associated SIDs. To enable the implementation of this registry, Section 4 defines a standard file format used to store and publish SIDs.

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- action
- feature

- module
- notification
- RPC
- schema node
- schema tree
- submodule

This specification also makes use of the following terminology:

- delta : Difference between the current SID and a reference SID. A reference SID is defined for each context for which deltas are used.
- item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- path: A path is a string that identifies a schema node within the schema tree. A path consists of the list of schema node identifier(s) separated by slashes ("/"). Schema node identifier(s) are always listed from the top-level schema node up to the targeted schema node. (e.g. "/ietf-system:system-state/clock/current-datetime")
- YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

3. ".sid" file lifecycle

YANG is a language designed to model data accessed using one of the compatible protocols (e.g. NETCONF [RFC6241], RESCONF [RFC8040] and CoMI [I-D.ietf-core-comi]). A YANG module defines hierarchies of data, including configuration, state data, RPCs, actions and notifications.

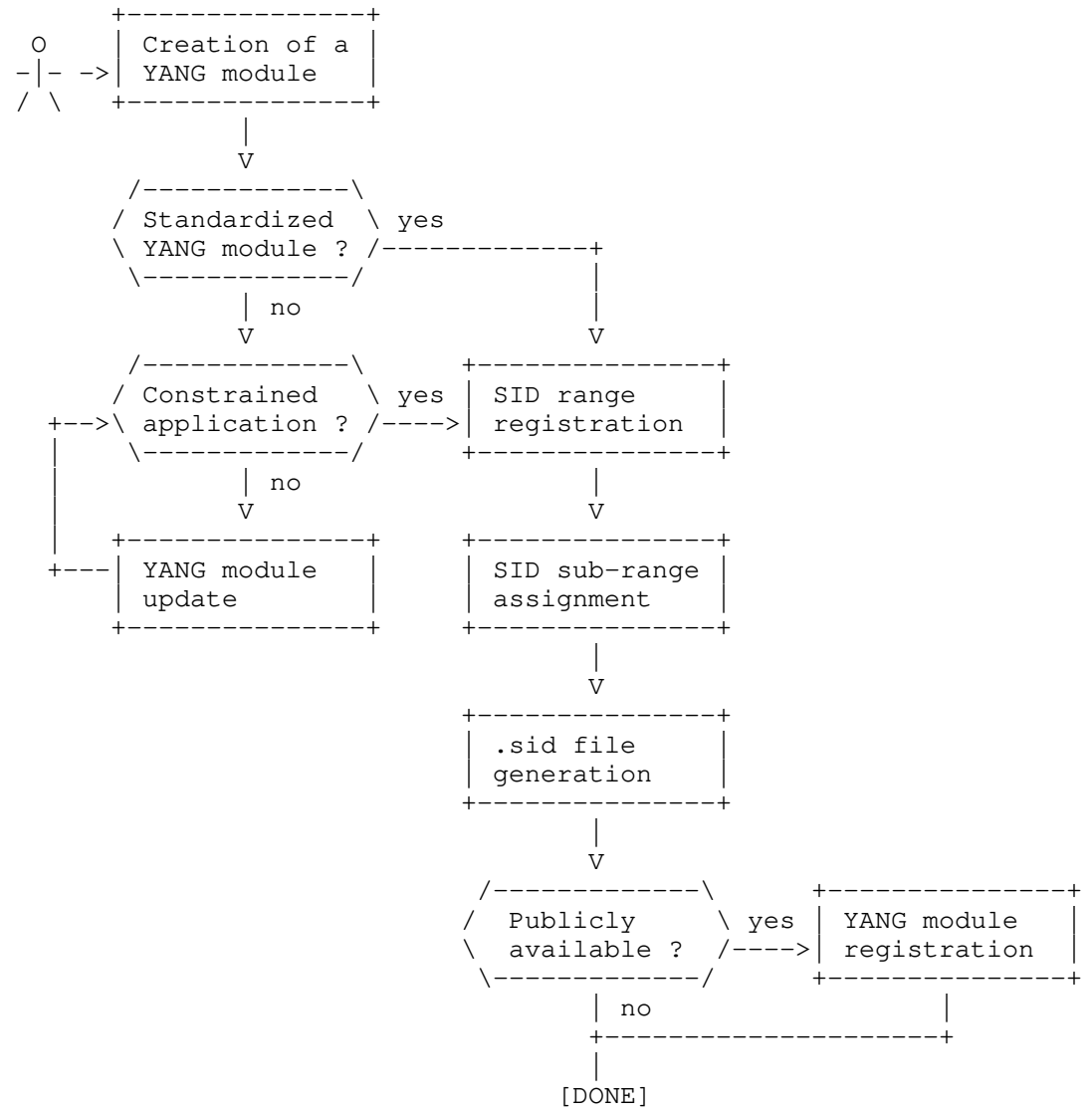
YANG modules are not necessary created in the context of constrained applications. YANG modules can be implemented using NETCONF [RFC6241] or RESTCONF [RFC8040] without the need to assign SIDs.

As needed, authors of YANG modules can assign SIDs to their YANG modules. This process starts by the registration of a SID range. Once a SID range is registered, the owner of this range assigns sub-ranges to each YANG module in order to generate the associated ".sid"

files. Generation of ".sid" files SHOULD be performed using an automated tool.

Registration of the .sid file associated to a YANG module is optional but recommended to promote interoperability between devices and to avoid duplicate allocation of SIDs to a single YANG module.

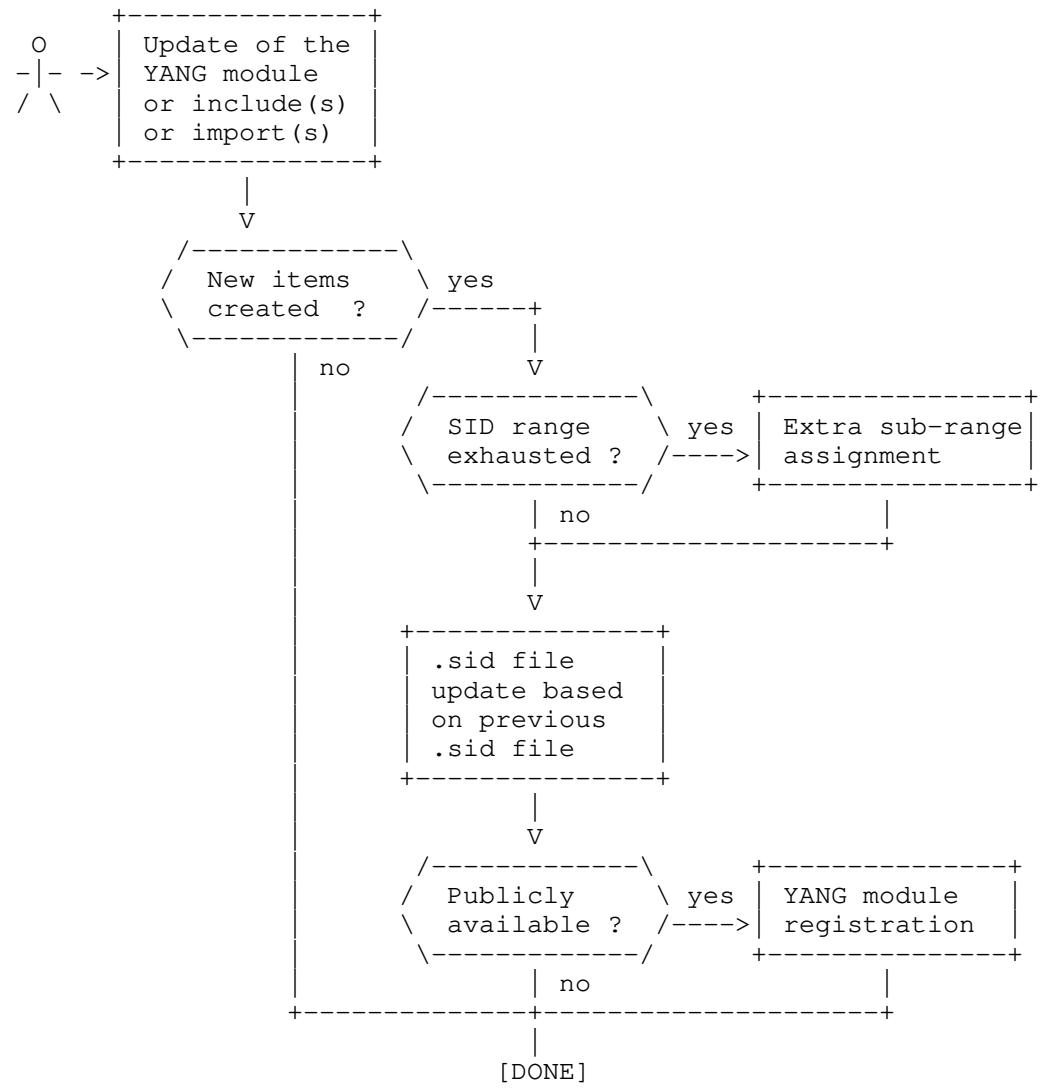
The following activity diagram summarizes the creation of a YANG module and its associated .sid file.



Each time a YANG module or one of its imported module(s) or included sub-module(s) is updated, the ".sid" file MAY need to be updated. This update SHOULD also be performed using an automated tool.

If a new revision requires more SIDs than initially allocated, a new SID range MUST be added to the 'assignment-ranges'. These extra SIDs are used for subsequent assignements.

The following activity diagram summarizes the update of a YANG module and its associated .sid file.



4. ".sid" file format

".sid" files are used to persist and publish SIDs assigned to the different YANG items of a specific YANG module. The following YANG module defined the structure of this file, encoding is performed using the rules defined in [RFC7951].

```
<CODE BEGINS> file "ietf-sid-file@2017-11-26.yang"
module ietf-sid-file {
```

```
namespace "urn:ietf:params:xml:ns:yang:ietf-sid-file";
prefix sid;

import ietf-yang-types {
  prefix yang;
}

import ietf-comi {
  prefix comi;
}

organization
  "IETF Core Working Group";

contact
  "Michel Veillette
   <mailto:michel.veillette@trilliantinc.com>

   Andy Bierman
   <mailto:andy@yumaworks.com>

   Alexander Pelov
   <mailto:a@ackl.io>";

description
  "This module defines the structure of the .sid files.

   Each .sid file contains the mapping between the different
   string identifiers defined by a YANG module and a
   corresponding numeric value called SID.";

revision 2017-11-26 {
  description
    "Initial revision.";
  reference
    "[I-D.ietf-core-sid] YANG Schema Item iDentifier (SID)";
}

typedef revision-identifier {
  type string {
    pattern '\d{4}-\d{2}-\d{2}';
  }
  description
    "Represents a date in YYYY-MM-DD format.";
}

typedef schema-node-path {
  type string {
```

```
        pattern
          '/[a-zA-Z_][a-zA-Z0-9\-_.]*:[a-zA-Z_][a-zA-Z0-9\-_.]*' +
          '(/[a-zA-Z_][a-zA-Z0-9\-_.]*(:[a-zA-Z_][a-zA-Z0-9\-_.]*)?)*';
      }
      description
        "Identifies a schema-node path string for use in the
         SID registry. This string format follows the rules
         for an instance-identifier, as defined in RFC 7959,
         except that no predicates are allowed.

         This format is intended to support the YANG 1.1 ABNF
         for a schema node identifier, except module names
         are used instead of prefixes, as specified in RFC 7951.";
      reference
        "RFC 7950, The YANG 1.1 Data Modeling Language;
         Section 6.5: Schema Node Identifier;
         RFC 7951, JSON Encoding of YANG Data;
         Section 6.11: The instance-identifier type";
    }

    leaf module-name {
      type yang:yang-identifier;
      description
        "Name of the YANG module associated with this .sid file.";
    }

    leaf module-revision {
      type revision-identifier;
      description
        "Revision of the YANG module associated with this .sid file.
         This leaf is not present if no revision statement is
         defined in the YANG module.";
    }

    list assigment-ranges {
      key "entry-point";
      description
        "SID range(s) allocated to the YANG module identified by
         'module-name' and 'module-revision'.";

      leaf entry-point {
        type comi:sid;
        mandatory true;
        description
          "Lowest SID available for assignment.";
      }

      leaf size {
```

```
        type uint64;
        mandatory true;
        description
          "Number of SIDs available for assignment.";
      }
    }

    list items {
      key "namespace identifier";
      description
        "Each entry within this list defined the mapping between
         a YANG item string identifier and a SID. This list MUST
         include a mapping entry for each YANG item defined by
         the YANG module identified by 'module-name' and
         'module-revision'.";

      leaf namespace {
        type enumeration {
          enum module {
            value 0;
            description
              "All module and submodule names share the same
               global module identifier namespace.";
          }
          enum identity {
            value 1;
            description
              "All identity names defined in a module and its
               submodules share the same identity identifier
               namespace.";
          }
          enum feature {
            value 2;
            description
              "All feature names defined in a module and its
               submodules share the same feature identifier
               namespace.";
          }
          enum data {
            value 3;
            description
              "The namespace for all data nodes, as defined in YANG.";
          }
        }
        description
          "Namespace of the YANG item for this mapping entry.";
      }
```

```
      leaf identifier {
        type union {
          type yang:yang-identifier;
          type schema-node-path;
        }
        description
          "String identifier of the YANG item for this mapping entry.

           If the corresponding 'namespace' field is 'module',
           'feature', or 'identity', then this field MUST
           contain a valid YANG identifier string.

           If the corresponding 'namespace' field is 'data',
           then this field MUST contain a valid schema node
           path.";
       }

      leaf sid {
        type comi:sid;
        mandatory true;
        description
          "SID assigned to the YANG item for this mapping entry.";
      }
    }
  }
  <CODE ENDS>
```

5. Security Considerations

The security considerations of [RFC7049] and [RFC7950] apply.

This document defines a new type of identifier used to encode data models defined in YANG [RFC7950]. As such, this identifier does not contribute to any new security issues in addition of those identified for the specific protocols or contexts for which it is used.

6. IANA Considerations

6.1. "SID mega-range" registry

The name of this registry is "SID mega-range". This registry is used to delegate the management of block of SIDs for third party's (e.g. SDO, registrar).

Each entry in this registry must include:

- The entry point (first entry) of the registered SID range.

- The size of the registered SID range.
- The contact information of the requesting organization including:
  - Organization name
  - Primary contact name, email address, and phone number
  - Secondary contact name, email address, and phone number

The initial entry in this registry is allocated to IANA:

Entry Point	Size	Organization name
0	1000000	IANA

The IANA policies for future additions to this registry are "Hierarchical Allocation, Expert Review" [RFC5226]. Prior to a first allocation, the requesting organization must demonstrate a functional registry infrastructure. On subsequent allocation request(s), the organization must demonstrate the exhaustion of the prior range. These conditions need to be asserted by the assigned expert(s).

6.1.1. IANA SID Mega-Range Registry

The first million SIDs assigned to IANA is sub-divided as follow:

- The range of 0 to 999 is reserved for future extensions. The IANA policy for this range is "IETF review" [RFC5226].
- The range of 1000 to 59,999 is reserved for YANG modules defined in RFCs. The IANA policy for future additions to this sub-registry is "RFC required" [RFC5226]. Allocation within this range requires publishing of the associated ".yang" and ".sid" files in the YANG module registry. The allocation within this range is done prior to the RFC publication but should not be done prior to the working group adoption.
- The range of 60,000 to 99,999 is reserved for experimental YANG modules. This range MUST NOT be used in operational deployments since these SIDs are not globally unique which limit their interoperability. The IANA policy for this range is "Experimental use" [RFC5226].
- The range of 100,000 to 999,999 is reserved for standardized YANG modules. The IANA policy for future additions to this sub-

registry is "Specification Required" [RFC5226]. Allocation within this range requires publishing of the associated ".yang" and ".sid" files in the YANG module registry.

Entry Point	Size	IANA policy
0	1,000	IETF review
1,000	59,000	RFC required
60,000	40,000	Experimental use
100,000	1,000,000,000	Specification Required

The size of a SID range assigned to a YANG module should be at least 33% above the current number of YANG items. This headroom allows assignment within the same range of new YANG items introduced by subsequent revisions. A larger SID range size may be requested by the authors if this recommendation is considered insufficient. It is important to note that an extra SID range can be allocated to an existing YANG module if the initial range is exhausted.

6.1.2. IANA "RFC SID range assignment" sub-registries

The name of this sub-registry is "RFC SID range assignment". This sub-registry corresponds to the SID entry point 1000, size 59000. Each entry in this sub-registry must include the SID range entry point, the SID range size, the YANG module name, the RFC number.

Initial entries in this registry are as follows:

Entry Point	Size	Module name	RFC number
1000	100	ietf-comi	[I-D.ietf-core-comi]
1100	50	ietf-yang-types	[RFC6021]
1150	50	ietf-inet-types	[RFC6021]
1200	50	iana-crypt-hash	[RFC7317]
1250	50	ietf-netconf-acm	[RFC6536]
1300	50	ietf-sid-file	RFCXXXX
1500	100	ietf-interfaces	[RFC7223]
1600	100	ietf-ip	[RFC7277]
1700	100	ietf-system	[RFC7317]
1800	400	iana-if-type	[RFC7224]

// RFC Ed.: replace XXXX with RFC number assigned to this draft.

6.2. "YANG module assignment" registry

The name of this registry is "YANG module assignment". This registry is used to track which YANG modules have been assigned and the specific YANG items assignment. Each entry in this sub-registry must include:

- The YANG module name
- The associated ".yang" file(s)
- The associated ".sid" file

The validity of the ".yang" and ".sid" files added to this registry MUST be verified.

- The syntax of the registered ".yang" and ".sid" files must be valid.
- Each YANG item defined by the registered ".yang" file must have a corresponding SID assigned in the ".sid" file.
- Each SID is assigned to a single YANG item, duplicate assignment is not allowed.
- The SID range(s) defined in the ".sid" file must be unique, must not conflict with any other SID ranges defined in already registered ".sid" files.
- The ownership of the SID range(s) should be verify.

The IANA policy for future additions to this registry is "First Come First Served" as described in [RFC5226].

7. Acknowledgments

The authors would like to thank Andy Bierman, Carsten Bormann, Abhinav Somaraju, Laurent Toutain and Randy Turner for their help during the development of this document and their useful comments during the review process.

8. References

8.1. Normative References

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

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

8.2. Informative References

[I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-01 (work in progress), July 2017.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <https://www.rfc-editor.org/info/rfc6021>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <https://www.rfc-editor.org/info/rfc6536>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <https://www.rfc-editor.org/info/rfc7224>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <https://www.rfc-editor.org/info/rfc7277>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

Appendix A. ".sid" file example

The following .sid file (ietf-system@2014-08-06.sid) have been generated using the following yang modules:

- ietf-system@2014-08-06.yang
- ietf-yang-types@2013-07-15.yang
- ietf-inet-types@2013-07-15.yang
- ietf-netconf-acm@2012-02-22.yang
- iana-crypt-hash@2014-04-04.yang


```
{
  "assignment-ranges": [
    {
      "entry-point": 1700,
      "size": 100
    }
  ],
  "module-name": "ietf-system",
  "module-revision": "2014-08-06",
  "items": [
    {
      "namespace": "module",
      "identifier": "ietf-system",
      "sid": 1700
    },
    {
      "namespace": "identity",
```

```
      "identifier": "authentication-method",
      "sid": 1701
    },
    {
      "namespace": "identity",
      "identifier": "local-users",
      "sid": 1702
    },
    {
      "namespace": "identity",
      "identifier": "radius",
      "sid": 1703
    },
    {
      "namespace": "identity",
      "identifier": "radius-authentication-type",
      "sid": 1704
    },
    {
      "namespace": "identity",
      "identifier": "radius-chap",
      "sid": 1705
    },
    {
      "namespace": "identity",
      "identifier": "radius-pap",
      "sid": 1706
    },
    {
      "namespace": "feature",
      "identifier": "authentication",
      "sid": 1707
    },
    {
      "namespace": "feature",
      "identifier": "dns-udp-tcp-port",
      "sid": 1708
    },
    {
      "namespace": "feature",
      "identifier": "local-users",
      "sid": 1709
    },
    {
      "namespace": "feature",
      "identifier": "ntp",
      "sid": 1710
    },
```

```
{
  "namespace": "feature",
  "identifier": "ntp-udp-port",
  "sid": 1711
},
{
  "namespace": "feature",
  "identifier": "radius",
  "sid": 1712
},
{
  "namespace": "feature",
  "identifier": "radius-authentication",
  "sid": 1713
},
{
  "namespace": "feature",
  "identifier": "timezone-name",
  "sid": 1714
},
{
  "namespace": "data",
  "identifier": "/ietf-system:set-current-datetime",
  "sid": 1715
},
{
  "namespace": "data",
  "identifier": "/ietf-system:set-current-datetime/
                 current-datetime",
  "sid": 1716
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system",
  "sid": 1717
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-restart",
  "sid": 1718
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-shutdown",
  "sid": 1719
},
{
  "namespace": "data",
```

```
        "identifier": "/ietf-system:system-state",
        "sid": 1720
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/clock",
        "sid": 1721
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/clock/boot-datetime",
        "sid": 1722
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/clock/
                       current-datetime",
        "sid": 1723
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform",
        "sid": 1724
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform/machine",
        "sid": 1725
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform/os-name",
        "sid": 1726
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform/os-release",
        "sid": 1727
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform/os-version",
        "sid": 1728
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/authentication",
        "sid": 1729
```

```
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user",
      "sid": 1730
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/
                     user-authentication-order",
      "sid": 1731
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     authorized-key",
      "sid": 1732
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     authorized-key/algorithm",
      "sid": 1733
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     authorized-key/key-data",
      "sid": 1734
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     authorized-key/name",
      "sid": 1735
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     name",
      "sid": 1736
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     password",
      "sid": 1737
    },
```

```
{
  "namespace": "data",
  "identifier": "/ietf-system:system/clock",
  "sid": 1738
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/clock/timezone-name",
  "sid": 1739
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/clock/timezone-utc-offset",
  "sid": 1740
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/contact",
  "sid": 1741
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver",
  "sid": 1742
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver/options",
  "sid": 1743
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver/options/
                 attempts",
  "sid": 1744
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver/options/
                 timeout",
  "sid": 1745
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver/search",
  "sid": 1746
},
{
```

```
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server",
      "sid": 1747
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/name",
      "sid": 1748
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/
                     udp-and-tcp",
      "sid": 1749
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/
                     udp-and-tcp/address",
      "sid": 1750
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/
                     udp-and-tcp/port",
      "sid": 1751
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/hostname",
      "sid": 1752
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/location",
      "sid": 1753
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp",
      "sid": 1754
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/enabled",
      "sid": 1755
    },
    {
```

```
        "namespace": "data",
        "identifier": "/ietf-system:system/ntp/server",
        "sid": 1756
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/ntp/server/
                       association-type",
        "sid": 1757
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/ntp/server/iburst",
        "sid": 1758
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/ntp/server/name",
        "sid": 1759
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/ntp/server/prefer",
        "sid": 1760
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/ntp/server/udp",
        "sid": 1761
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/ntp/server/udp/address",
        "sid": 1762
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/ntp/server/udp/port",
        "sid": 1763
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/radius",
        "sid": 1764
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system/radius/options",
```

```
      "sid": 1765
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/options/attempts",
      "sid": 1766
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/options/timeout",
      "sid": 1767
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server",
      "sid": 1768
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/
                     authentication-type",
      "sid": 1769
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/name",
      "sid": 1770
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/udp",
      "sid": 1771
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/udp/
                     address",
      "sid": 1772
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/udp/
                     authentication-port",
      "sid": 1773
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/udp/
```

```
                        shared-secret",
          "sid": 1774
        }
      ]
    }
```

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov (editor)
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: August 11, 2018

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
A. Somaraju
Tridonic GmbH & Co KG
R. Turner
Landis+Gyr
A. Minaburo
Acklio

February 07, 2018

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-06

Abstract

This document defines encoding rules for serializing configuration data, state data, RPC input and RPC output, Action input, Action output and notifications defined within YANG modules using the Concise Binary Object Representation (CBOR) [RFC7049].

Status of This Memo

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

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

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

This Internet-Draft will expire on August 11, 2018.

Table of Contents

1. Introduction 3
2. Terminology and Notation 3
2.1. YANG Schema Item iDentifier (SID) 5
2.2. CBOR diagnostic notation 5
3. Properties of the CBOR Encoding 6
4. Encoding of YANG Data Node Instances 7
4.1. The 'leaf' Data Node 7
4.2. The 'container' Data Node 7
4.2.1. SIDs as keys 8
4.2.2. Member names as keys 10
4.3. The 'leaf-list' Data Node 10
4.4. The 'list' Data Node 11
4.4.1. SIDs as keys 11
4.4.2. Member names as keys 14
4.5. The 'anydata' Data Node 15
4.6. The 'anyxml' Data Node 17
5. Encoding of YANG data templates 17
5.1. SIDs as keys 18
5.2. Member names as keys 19
6. Representing YANG Data Types in CBOR 20
6.1. The unsigned integer Types 20
6.2. The integer Types 21
6.3. The 'decimal64' Type 21
6.4. The 'string' Type 22
6.5. The 'boolean' Type 22
6.6. The 'enumeration' Type 22
6.7. The 'bits' Type 23
6.8. The 'binary' Type 24
6.9. The 'leafref' Type 24
6.10. The 'identityref' Type 25
6.10.1. SIDs as identityref 25
6.10.2. Name as identityref 26
6.11. The 'empty' Type 26
6.12. The 'union' Type 27
6.13. The 'instance-identifier' Type 28
6.13.1. SIDs as instance-identifier 28
6.13.2. Names as instance-identifier 31
7. Security Considerations 32
8. IANA Considerations 32

8.1. Tags Registry 32
9. Acknowledgments 32
10. References 33
10.1. Normative References 33
10.2. Informative References 33
Authors' Addresses 34

1. Introduction

The specification of the YANG 1.1 data modelling language [RFC7950] defines an XML encoding for data instances, i.e. contents of configuration datastores, state data, RPC inputs and outputs, action inputs and outputs, and event notifications.

A new set of encoding rules has been defined to allow the use of the same data models in environments based on the JavaScript Object Notation (JSON) Data Interchange Format [RFC7159]. This is accomplished in the JSON Encoding of Data Modeled with YANG specification [RFC7951].

The aim of this document is to define a set of encoding rules for the Concise Binary Object Representation (CBOR) [RFC7049]. The resulting encoding is more compact compared to XML and JSON and more suitable for Constrained Nodes and/or Constrained Networks as defined by [RFC7228].

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- action
- anydata
- anyxml
- data node
- data tree
- datastore
- feature

- identity
- module
- notification
- RPC
- schema node
- schema tree
- submodule

The following terms are defined in [RFC7951]:

- member name
- name of an identity
- namespace-qualified

The following terms are defined in [RFC8040]:

- yang-data (YANG extension)
- YANG data template

This specification also makes use of the following terminology:

- child: A schema node defined within a collection such as a container, a list, a case, a notification, an RPC input, an RPC output, an action input, an action output.
- delta: Difference between the current SID and a reference SID. A reference SID is defined for each context for which deltas are used.
- item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- parent: The collection in which a schema node is defined.
- YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

2.1. YANG Schema Item iDentifier (SID)

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called YANG Schema Item iDentifier (SID), is encoded using an unsigned integer. The following items are identified using SIDs:

- identities
- data nodes
- RPCs and associated input(s) and output(s)
- actions and associated input(s) and output(s)
- notifications and associated information
- YANG modules, submodules and features

To minimize its size, in certain positions, SIDs are represented using a (signed) delta from a reference SID and the current SID. Conversion from SIDs to deltas and back to SIDs are stateless processes solely based on the data serialized or deserialized.

Mechanisms and processes used to assign SIDs to YANG items and to guarantee their uniqueness is outside the scope of the present specification. If SIDs are to be used, the present specification is used in conjunction with a specification defining this management. One example for such a specification is under development as [I-D.ietf-core-sid].

2.2. CBOR diagnostic notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation as defined in [RFC7049] section 6. This notation is used strictly for documentation purposes and is never used in the data serialization. Table 1 below provides a summary of this notation.

CBOR content	CBOR type	Diagnostic notation	Example	CBOR encoding
Unsigned integer	0	Decimal digits	123	18 7b
Negative integer	1	Decimal digits prefixed by a minus sign	-123	38 7a
Byte string	2	Hexadecimal value enclosed between single quotes and prefixed by an 'h'	h'f15c'	42 f15c
Text string	3	String of Unicode characters enclosed between double quotes	"txt"	63 747874
Array	4	Comma-separated list of values within square brackets	[1, 2]	82 01 02
Map	5	Comma-separated list of key : value pairs within curly braces	{ 1: 123, 2: 456 }	a2 01187b 021901c8
Boolean	7/20	false	false	f4
	7/21	true	true	f5
Null	7/22	null	null	f6
Not assigned	7/23	undefined	undefined	f7

Table 1: CBOR diagnostic notation summary

The following extensions to the CBOR diagnostic notation are supported:

- Any text within and including a pair of slashes is considered a comment.

- Deltas are visualized as numbers preceded by a '+' or '-' sign. The use of the '+' sign for positive deltas represents an extension to the CBOR diagnostic notation as defined by [RFC7049] section 6.

3. Properties of the CBOR Encoding

This document defines CBOR encoding rules for YANG schema trees and their subtrees.

Basic schema nodes such as leaf, leaf-list, list, anydata and anyxml can be encoded standalone. In this case, only the value of this

schema node is encoded in CBOR. Identification of this value needs to be provided by some external means when required.

A collection such as container, list instance, notification, RPC input, RPC output, action input and action output is serialized using a CBOR map in which each child schema node is encoded using a key and a value. This specification supports two type of CBOR keys; YANG Schema Item iDentifier (SID) as defined in Section 2.1 and member names as defined in [RFC7951]. Each of these key types is encoded using a specific CBOR type which allows their interpretation during the deserialization process. The end user of this mapping specification (e.g. RESTCONF [RFC8040], CoMI [I-D.ietf-core-comi]) can mandate the use of a specific key type.

In order to minimize the size of the encoded data, the proposed mapping avoids any unnecessary meta-information beyond those natively supported by CBOR. For instance, CBOR tags are used solely in the case of anyxml data nodes and the union datatype to distinguish explicitly the use of different YANG datatypes encoded using the same CBOR major type.

4. Encoding of YANG Data Node Instances

Schema node instances defined using the YANG modeling language are encoded using CBOR [RFC7049] based on the rules defined in this section. We assume that the reader is already familiar with both YANG [RFC7950] and CBOR [RFC7049].

4.1. The 'leaf' Data Node

Leafs MUST be encoded based on the encoding rules specified in Section 6.

4.2. The 'container' Data Node

Collections such as containers, list instances, notifications, RPC inputs, RPC outputs, action inputs and action outputs MUST be encoded using a CBOR map data item (major type 5). A map is comprised of pairs of data items, with each data item consisting of a key and a value. Each key within the CBOR map is set to a data node identifier, each value is set to the value of this data node instance according to the instance datatype.

This specification supports two type of CBOR keys; SID as defined in Section 2.1 encoded as deltas and member names as defined in [RFC7951] encoded using CBOR text strings. The use of CBOR byte strings for keys is reserved for future extensions.

4.2.1. SIDs as keys

Keys implemented using SIDs MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value. Keys are represented as the delta of the associated SID, delta values are computed as follows:

- The delta value is equal to the SID of the current schema node minus the SID of the parent schema node. When no parent exists in the context of use of this container, the delta is set to the SID of the current schema node (i.e., a parent with SID equal to zero is assumed).

- Delta values may result in a negative number, clients and servers MUST support both unsigned and negative deltas.

The following example shows the encoding of a 'system-state' container instance with a single child, a clock container. The clock container container has two children, a 'current-datetime' leaf and a 'boot-datetime' leaf.

Definition example from [RFC7317]:

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
             \d{2}:\d{2})';
  }
}

container system-state {

  container clock {
    leaf current-datetime {
      type date-and-time;
    }

    leaf boot-datetime {
      type date-and-time;
    }
  }
}
```

For this first representation, we assume that the SID of the parent container (i.e. 'system-state') is not available to the serializer. In this case, root data nodes are encoded using absolute SIDs.

CBOR diagnostic notation:

```
{
  1717 : {                               / clock  (SID 1717) /
    +2 : "2015-10-02T14:47:24Z-05:00", / current-datetime (SID 1719)/
    +1 : "2015-09-15T09:12:58Z-05:00"  / boot-datetime (SID 1718) /
  }
}
```

CBOR encoding:

```
a1                                     # map(1)
   19 06b5                             # unsigned(1717)
   a2                                  # map(2)
      02                               # unsigned(2)
      78 1a                            # text(26)
      323031352d31302d30325431343a34373a32345a2d30353a3030
      01                               # unsigned(1)
      78 1a                            # text(26)
      323031352d30392d31355430393a31323a35385a2d30353a3030
```

On the other hand, if the serializer is aware of the parent SID, 1716 in the case 'system-state' container, root data nodes are encoded using deltas.

CBOR diagnostic notation:

```
{
  +1 : {                               / clock  (SID 1717) /
    +2 : "2015-10-02T14:47:24Z-05:00", / current-datetime (SID 1719)/
    +1 : "2015-09-15T09:12:58Z-05:00"  / boot-datetime (SID 1718) /
  }
}
```

CBOR encoding:

```
a1                                     # map(1)
   01                                   # unsigned(1)
   a2                                  # map(2)
      02                               # unsigned(2)
      78 1a                            # text(26)
      323031352d31302d30325431343a34373a32345a2d30353a3030
      01                               # unsigned(1)
      78 1a                            # text(26)
      323031352d30392d31355430393a31323a35385a2d30353a3030
```

4.2.2. Member names as keys

Keys implemented using member names MUST be encoded using a CBOR text string data item (major type 3). A namespace-qualified member name MUST be used for all members of a top-level collection, and then also whenever the namespaces of the schema node and its parent are different. In all other cases, the simple form of the member name MUST be used. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of a 'system' container instance using names. This example is described in Section 4.2.1.

CBOR diagnostic notation:

```
{
  "ietf-system:clock" : {
    "current-datetime" : "2015-10-02T14:47:24Z-05:00",
    "boot-datetime" : "2015-09-15T09:12:58Z-05:00"
  }
}
```

CBOR encoding:

```
a1                                          # map(1)
   71                                       # text(17)
      696574662d73797374656d3a636c6f636b    # "ietf-system:clock"
   a2                                       # map(2)
      70                                    # text(16)
         63757272656e742d6461746574696d65   # "current-datetime"
      78 1a                                 # text(26)
         323031352d31302d30325431343a34373a32345a2d30353a3030
      6d                                    # text(13)
         626f6f742d6461746574696d65         # "boot-datetime"
      78 1a                                 # text(26)
         323031352d30392d31355430393a31323a35385a2d30353a3030
```

4.3. The 'leaf-list' Data Node

A leaf-list MUST be encoded using a CBOR array data item (major type 4). Each entry of this array MUST be encoded using the rules defined by the YANG type specified.

The following example shows the encoding a 'search' leaf-list instance containing the two entries, "ietf.org" and "ieee.org".

Definition example [RFC7317]:

```
typedef domain-name {
  type string {
    length "1..253";
    pattern '((([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9].)
             *([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.?
             )|\.';
  }
}

leaf-list search {
  type domain-name;
  ordered-by user;
}
```

CBOR diagnostic notation: ["ietf.org", "ieee.org"]

CBOR encoding: 82 68 696574662e6f7267 68 696565652e6f7267

4.4. The 'list' Data Node

A list MUST be encoded using a CBOR array data item (major type 4). Each list instance within this CBOR array is encoded using a CBOR map data item (major type 5) based on the same rules as a YANG container as defined in Section 4.2.

4.4.1. SIDs as keys

The following example show the encoding of a 'server' list instance using SIDs. It is important to note that the protocol or method using this mapping may carry a parent SID or may have the knowledge of this parent SID based on its context. In these cases, delta encoding can be performed based on this parent SID which minimizes the size of the encoded data.

Definition example from [RFC7317]:

```
   list server {
     key name;

     leaf name {
       type string;
     }
     choice transport {
       case udp {
         container udp {
           leaf address {
             type host;
             mandatory true;
           }
           leaf port {
             type port-number;
           }
         }
       }
     }
     leaf association-type {
       type enumeration {
         enum server;
         enum peer;
         enum pool;
       }
       default server;
     }
     leaf iburst {
       type boolean;
       default false;
     }
     leaf prefer {
       type boolean;
       default false;
     }
   }
```

CBOR diagnostic notation:

```
[
  {
    1755 : "NRC TIC server",          / name (SID 1755) /
    1757 : {                          / udp (SID 1757) /
      +1 : "tic.nrc.ca",              / address (SID 1758) /
      +2 : 123                        / port (SID 1759) /
    },
    1753 : 0,                         / association-type (SID 1753) /
    1754 : false,                     / iburst (SID 1754) /
    1756 : true                       / prefer (SID 1756) /
  },
  {
    1755 : "NRC TAC server",          / name (SID 1755) /
    1757 : {                          / udp (SID 1757) /
      +1 : "tac.nrc.ca"               / address (SID 1758) /
    }
  }
]
```

CBOR encoding:

```
82                                    # array(2)
   a5                                 # map(5)
      19 06db                         # unsigned(1755)
      6e                              # text(14)
         4e524320544943207365727672657 # "NRC TIC server"
      19 06dd                         # unsigned(1757)
      a2                              # map(2)
         01                           # unsigned(1)
         6a                           # text(10)
            7469632e6e72632e6361      # "tic.nrc.ca"
         02                           # unsigned(2)
         18 7b                        # unsigned(123)
      19 06d9                         # unsigned(1753)
      00                              # unsigned(0)
      19 06da                         # unsigned(1754)
      f4                              # primitive(20)
      19 06dc                         # unsigned(1756)
      f5                              # primitive(21)
   a2                                 # map(2)
      19 06db                         # unsigned(1755)
      6e                              # text(14)
         4e524320544143207365727672657 # "NRC TAC server"
      19 06dd                         # unsigned(1757)
      a1                              # map(1)
         01                           # unsigned(1)
         6a                           # text(10)
            7461632e6e72632e6361      # "tac.nrc.ca"
```

4.4.2. Member names as keys

The following example shows the encoding of a 'server' list instance using names. This example is described in Section 4.4.1.

CBOR diagnostic notation:

```
[
  {
    "ietf-system:name" : "NRC TIC server",
    "ietf-system:udp" : {
      "address" : "tic.nrc.ca",
      "port" : 123
    },
    "ietf-system:association-type" : 0,
    "ietf-system:iburst" : false,
    "ietf-system:prefer" : true
  },
  {
    "ietf-system:name" : "NRC TAC server",
    "ietf-system:udp" : {
      "address" : "tac.nrc.ca"
    }
  }
]
```

CBOR encoding:

```
82                                              # array(2)
   a5                                           # map(5)
      70                                        # text(16)
         696574662d73797374656d3a6e616d65       # "ietf-system:name"
      6e                                        # text(14)
         4e524320544943207365727672657 2        # "NRC TIC server"
      6f                                        # text(15)
         696574662d73797374656d3a756470         # "ietf-system:udp"
      a2                                        # map(2)
         67                                     # text(7)
            61646472657373                      # "address"
         6a                                     # text(10)
            7469632e6e72632e6361                # "tic.nrc.ca"
         64                                     # text(4)
            706f7274                            # "port"
         18 7b                                  # unsigned(123)
      78 1c                                     # text(28)
         696574662d73797374656d3a6173736f63696174696f6e2d74797065
      00                                        # unsigned(0)
      72                                        # text(18)
         696574662d73797374656d3a696275727374 # "ietf-system:iburst"
      f4                                        # primitive(20)
      72                                        # text(18)
         696574662d73797374656d3a707265666572 # "ietf-system:prefer"
      f5                                        # primitive(21)
   a2                                           # map(2)
      70                                        # text(16)
         696574662d73797374656d3a6e616d65       # "ietf-system:name"
      6e                                        # text(14)
         4e524320544143207365727672657 2        # "NRC TAC server"
      6f                                        # text(15)
         696574662d73797374656d3a756470         # "ietf-system:udp"
      a1                                        # map(1)
         67                                     # text(7)
            61646472657373                      # "address"
         6a                                     # text(10)
            7461632e6e72632e6361                # "tac.nrc.ca"
```

4.5. The 'anydata' Data Node

An anydata serves as a container for an arbitrary set of schema nodes that otherwise appear as normal YANG-modeled data. An anydata instance is encoded using the same rules as a container, i.e., CBOR map. The requirement that anydata content can be modeled by YANG implies the following:

- Keys of any inner data nodes MUST be set to valid deltas or member names.

- The CBOR array MUST contain either unique scalar values (as a leaf-list, see Section 4.3), or maps (as a list, see Section 4.4).

- Values MUST follow the encoding rules of one of the datatypes listed in Section 6.

The following example shows a possible use of anydata. In this example, an anydata is used to define a data node containing a notification event, this data node can be part of a YANG list to create an event logger.

Definition example:

```
anydata event;
```

This example also assumes the assistance of the following notification.

```
module example-port {
  ...

  notification example-port-fault {  # SID 2600
    leaf port-name {                 # SID 2601
      type string;
    }
    leaf port-fault {                # SID 2601
      type string;
    }
  }
}
```

CBOR diagnostic notation:

```
{
  2601 : "0/4/21",        / port-name /
  2602 : "Open pin 2"     / port-fault /
}
```

CBOR encoding:

```
a2                          # map(2)
   19 0a29                  # unsigned(2601)
   66                       # text(6)
      302f342f3231          # "0/4/21"
   19 0a2a                  # unsigned(2602)
   6a                       # text(10)
      4f70656e2070696e2032  # "Open pin 2"
```

4.6. The 'anyxml' Data Node

An anyxml schema node is used to serialize an arbitrary CBOR content, i.e., its value can be any CBOR binary object. anyxml value may contain CBOR data items tagged with one of the tag listed in Section 8.1, these tags shall be supported.

The following example shows a valid CBOR encoded instance.

Definition example from [RFC7951]:

anyxml bar;

CBOR diagnostic notation: [true, null, true]

CBOR encoding: 83 f5 f6 f5

5. Encoding of YANG data templates

YANG data templates are data structures defined in YANG but not intended to be implemented as part of a datastore. YANG data templates are defined using the 'yang-data' extension as described by RFC 8040.

The encoding rules defined for YANG containers in section 4.2 may be used to serialize YANG data templates.

Definition example from [I-D.ietf-core-comi]:

```
import ietf-restconf {
  prefix rc;
}

rc:yang-data yang-errors {
  container error {
    leaf error-tag {
      type identityref {
        base error-tag;
      }
    }
    leaf error-app-tag {
      type identityref {
        base error-app-tag;
      }
    }
    leaf error-data-node {
      type instance-identifier;
    }
    leaf error-message {
      type string;
    }
  }
}
```

Just like YANG containers, YANG data templates can be encoded using either SIDs or names.

5.1. SIDs as keys

This example shows a serialization example of the yang-errors template using SIDs as CBOR map key.

CBOR diagnostic notation:

```
{
  1024 : {                    / error  (SID 1024) /
    +4 : 1011,                / error-tag (SID 1028) /
                              / = invalid-value (SID 1011) /
    +1 : 1018,                / error-app-tag (SID 1025) /
                              / = not-in-range (SID 1018) /
    +2 : 1740,                / error-data-node (SID 1026) /
                              / = timezone-utc-offset (SID 1740) /
    +3 : "max value exceeded" / error-message (SID 1027) /
  }
}
```

CBOR encoding:

```
A1                                      # map(1)
   19 0400                              # unsigned(1024)
   A4                                   # map(4)
      04                                # unsigned(4)
      19 03F3                           # unsigned(1011)
      01                                # unsigned(1)
      19 03FA                           # unsigned(1018)
      02                                # unsigned(2)
      19 06CC                           # unsigned(1740)
      03                                # unsigned(3)
      76                                # text(22)
         6D6178696D756D2076616C756520657863656565646564
```

5.2. Member names as keys

This example shows a serialization example of the yang-errors template using member names as CBOR map key.

CBOR diagnostic notation:

```
{
  "ietf-comi:error" : {
    "error-tag" : "invalid-value",
    "error-app-tag" : "not-in-range",
    "error-data-node" : "timezone-utc-offset",
    "error-message" : "max value exceeded"
  }
}
```

CBOR encoding:

```
A1                                      # map(1)
   6F                                   # text(15)
      696574662D636F6D693A6572726F72
   A4                                   # map(4)
      69                                # text(9)
         6572726F722D746167             # "error-tag"
      6D                                # text(13)
         696E76616C69642D76616C7565
      6D                                # text(13)
         6572726F722D6170702D746167
      6C                                # text(12)
         6E6F742D696E2D72616E6765
      6F                                # text(15)
         6572726F722D646174612D6E6F6465
      73                                # text(19)
         74696D657A6F6E652D7574632D6F6666736574
      6D                                # text(13)
         6572726F722D6D657373616765
      72                                # text(18)
         6D61782076616C7565206578636565646564
```

6. Representing YANG Data Types in CBOR

The CBOR encoding of an instance of a leaf or leaf-list data node depends on the built-in type of that data node. The following sub-section defined the CBOR encoding of each built-in type supported by YANG as listed in [RFC7950] section 4.2.4. Each subsection shows an example value assigned to a data node instance of the discussed built-in type.

6.1. The unsigned integer Types

Leafs of type uint8, uint16, uint32 and uint64 MUST be encoded using a CBOR unsigned integer data item (major type 0).

The following example shows the encoding of a ’mtu’ leaf instance set to 1280 bytes.

Definition example from [RFC7277]:

```
leaf mtu {
  type uint16 {
    range "68..max";
  }
}
```

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

6.2. The integer Types

Leafs of type int8, int16, int32 and int64 MUST be encoded using either CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value.

The following example shows the encoding of a 'timezone-utc-offset' leaf instance set to -300 minutes.

Definition example from [RFC7317]:

```
leaf timezone-utc-offset {
  type int16 {
    range "-1500 .. 1500";
  }
}
```

CBOR diagnostic notation: -300

CBOR encoding: 39 012b

6.3. The 'decimal64' Type

Leafs of type decimal64 MUST be encoded using a decimal fraction as defined in [RFC7049] section 2.4.3.

The following example shows the encoding of a 'my-decimal' leaf instance set to 2.57.

Definition example from [RFC7317]:

```
leaf my-decimal {
  type decimal64 {
    fraction-digits 2;
    range "1 .. 3.14 | 10 | 20..max";
  }
}
```

CBOR diagnostic notation: 4([-2, 257])

CBOR encoding: c4 82 21 19 0101

6.4. The 'string' Type

Leafs of type string MUST be encoded using a CBOR text string data item (major type 3).

The following example shows the encoding of a 'name' leaf instance set to "eth0".

Definition example from [RFC7223]:

```
leaf name {
  type string;
}
```

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

6.5. The 'boolean' Type

Leafs of type boolean MUST be encoded using a CBOR true (major type 7, additional information 21) or false data item (major type 7, additional information 20).

The following example shows the encoding of an 'enabled' leaf instance set to 'true'.

Definition example from [RFC7317]:

```
leaf enabled {
  type boolean;
}
```

CBOR diagnostic notation: true

CBOR encoding: f5

6.6. The 'enumeration' Type

Leafs of type enumeration MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value. Enumeration values are either explicitly assigned using the YANG statement 'value' or automatically assigned based on the algorithm defined in [RFC7950] section 9.6.4.2.

The following example shows the encoding of an 'oper-status' leaf instance set to 'testing'.

Definition example from [RFC7317]:

```
leaf oper-status {
  type enumeration {
    enum up { value 1; }
    enum down { value 2; }
    enum testing { value 3; }
    enum unknown { value 4; }
    enum dormant { value 5; }
    enum not-present { value 6; }
    enum lower-layer-down { value 7; }
  }
}
```

CBOR diagnostic notation: 3

CBOR encoding: 03

6.7. The 'bits' Type

Leafs of type bits MUST be encoded using a CBOR byte string data item (major type 2). Bits position are either explicitly assigned using the YANG statement 'position' or automatically assigned based on the algorithm defined in [RFC7950] section 9.7.4.2.

Bits position 0 to 7 are assigned to the first byte within the byte string, bits 8 to 15 to the second byte, and subsequent bytes are assigned similarly. Within each byte, bits are assigned from least to most significant.

The following example shows the encoding of a 'mybits' leaf instance with the 'disable-nagle' and '10-Mb-only' flags set.

Definition example from [RFC7950]:

```
leaf mybits {
  type bits {
    bit disable-nagle {
      position 0;
    }
    bit auto-sense-speed {
      position 1;
    }
    bit 10-Mb-only {
      position 2;
    }
  }
}
```

CBOR diagnostic notation: h'05'

CBOR encoding: 41 05

6.8. The 'binary' Type

Leafs of type binary MUST be encoded using a CBOR byte string data item (major type 2).

The following example shows the encoding of an 'aes128-key' leaf instance set to 0x1f1ce6a3f42660d888d92a4d8030476e.

Definition example:

```
leaf aes128-key {
  type binary {
    length 16;
  }
}
```

CBOR diagnostic notation: h'1f1ce6a3f42660d888d92a4d8030476e'

CBOR encoding: 50 1f1ce6a3f42660d888d92a4d8030476e

6.9. The 'leafref' Type

Leafs of type leafref MUST be encoded using the rules of the schema node referenced by the 'path' YANG statement.

The following example shows the encoding of an 'interface-state-ref' leaf instance set to "eth1".

Definition example from [RFC7223]:

```
typedef interface-state-ref {
  type leafref {
    path "/interfaces-state/interface/name";
  }
}

container interfaces-state {
  list interface {
    key "name";
    leaf name {
      type string;
    }
    leaf-list higher-layer-if {
      type interface-state-ref;
    }
  }
}
```

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

6.10. The 'identityref' Type

This specification supports two approaches for encoding identityref, a YANG Schema Item iDentifier (SID) as defined in Section 2.1 or a name as defined in [RFC7951] section 6.8.

6.10.1. SIDs as identityref

When schema nodes of type identityref are implemented using SIDs, they MUST be encoded using a CBOR unsigned integer data item (major type 0). (Note that no delta mechanism is employed for SIDs as identityref.)

The following example shows the encoding of a 'type' leaf instance set to the value 'iana-if-type:ethernetCsmacd' (SID 1180).

Definition example from [RFC7317]:

```
identity interface-type {
}

identity iana-interface-type {
  base interface-type;
}

identity ethernetCsmacd {
  base iana-interface-type;
}

leaf type {
  type identityref {
    base interface-type;
  }
}
```

CBOR diagnostic notation: 1180

CBOR encoding: 19 049c

6.10.2. Name as identityref

Alternatively, an identityref may be encoded using a name as defined in [RFC7951] section 6.8. When names are used, identityref MUST be encoded using a CBOR text string data item (major type 3). If the identity is defined in another module than the leaf node containing the identityref value, the namespace-qualified form MUST be used. Otherwise, both the simple and namespace-qualified forms are permitted. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of the identity 'iana-if-type:ethernetCsmacd' using its name. This example is described in Section 6.10.1.

CBOR diagnostic notation: "iana-if-type:ethernetCsmacd"

CBOR encoding: 78 1b
69616e612d69662d747970653a65746865726e657443736d616364

6.11. The 'empty' Type

Leafs of type empty MUST be encoded using the CBOR null value (major type 7, additional information 22).

The following example shows the encoding of a 'is-router' leaf instance when present.

Definition example from [RFC7277]:

```
leaf is-router {
  type empty;
}
```

CBOR diagnostic notation: null

CBOR encoding: f6

6.12. The 'union' Type

Leafs of type union MUST be encoded using the rules associated with one of the types listed. When used in a union, the following YANG datatypes are prefixed by CBOR tag to avoid confusion between different YANG datatypes encoded using the same CBOR major type.

- bits
- enumeration
- identityref
- instance-identifier

See Section 8.1 for more information about these CBOR tags.

The following example shows the encoding of an 'ip-address' leaf instance when set to "2001:db8:a0b:12f0::1".

Definition example from [RFC7317]:

```
typedef ipv4-address {
  type string {
  pattern '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}
           ([0-9][1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(%[\p{N}
           \p{L}]+)?';
  }
}

typedef ipv6-address {
  type string {
    pattern '((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}((([0-9a
             -fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|(((25[0-5]|2[0-4][0
             -9]|[01]?[0-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0
             -9]?[0-9])))(%[\p{N}\p{L}]+)?';
    pattern '(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*))|((([^:]+:)*[^:]+)
             ?::(([^:]+:)*[^:]+)?)(%.+)?';
  }
}

typedef ip-address {
  type union {
    type ipv4-address;
    type ipv6-address;
  }
}

leaf address {
  type inet:ip-address;
}
```

CBOR diagnostic notation: "2001:db8:a0b:12f0::1"

CBOR encoding: 74 323030313a6462383a6130623a313266303a3a31

6.13. The 'instance-identifier' Type

This specification supports two approaches for encoding an instance-identifier, one based on YANG Schema Item iDentifier (SID) as defined in Section 2.1 and one based on names as defined in [RFC7951] section 6.11.

6.13.1. SIDs as instance-identifier

SIDs uniquely identify a data node. In the case of a single instance data node, a data node defined at the root of a YANG module or submodule or data nodes defined within a container, the SID is sufficient to identify this instance.

In the case of a data node member of a YANG list, a SID is combined with the list key(s) to identify each instance within the YANG list(s).

Single instance data nodes MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.

Data nodes member of a YANG list MUST be encoded using a CBOR array data item (major type 4) containing the following entries:

- The first entry MUST be encoded as a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.

- The following entries MUST contain the value of each key required to identify the instance of the targeted data node. These keys MUST be ordered as defined in the 'key' YANG statement, starting from top level list, and follow by each of the subordinate list(s).

First example:

The following example shows the encoding of a leaf instance of type instance-identifier which identifies the data node "/system/contact" (SID 1737).

Definition example from [RFC7317]:

```
container system {

  leaf contact {
    type string;
  }

  leaf hostname {
    type inet:domain-name;
  }
}
```

CBOR diagnostic notation: 1737

CBOR encoding: 19 06c9

Second example:

The following example shows the encoding of a leaf instance of type instance-identifier which identify the data node instance

"/system/authentication/user/authorized-key/key-data" (SID 1730) for user name "bob" and authorized-key "admin".

Definition example from [RFC7317]:

```
list user {
  key name;

  leaf name {
    type string;
  }
  leaf password {
    type ianach:crypt-hash;
  }

  list authorized-key {
    key name;

    leaf name {
      type string;
    }
    leaf algorithm {
      type string;
    }
    leaf key-data {
      type binary;
  }
}
```

CBOR diagnostic notation: [1730, "bob", "admin"]

CBOR encoding:

```
83                      # array(3)
   19 06c2              # unsigned(1730)
   63                   # text(3)
      626f62            # "bob"
   65                   # text(5)
      61646d696e        # "admin"
```

Third example:

The following example shows the encoding of a leaf instance of type instance-identifier which identify the list instance "/system/authentication/user" (SID 1726) corresponding to the user name "jack".

CBOR diagnostic notation: [1726, "jack"]

CBOR encoding:

```
82                      # array(2)
   19 06be              # unsigned(1726)
   64                   # text(4)
      6a61636b          # "jack"
```

6.13.2. Names as instance-identifier

The use of names as instance-identifier is defined in [RFC7951] section 6.11. The resulting xpath MUST be encoded using a CBOR text string data item (major type 3).

First example:

This example is described in Section 6.13.1.

CBOR diagnostic notation: "/ietf-system:system/contact"

CBOR encoding:

78 1c 2f20696574662d73797374656d3a73797374656d2f636f6e74616374

Second example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name='bob']/authorized-key [name='admin']/key-data"

CBOR encoding:

```
78 59
   2f696574662d73797374656d3a73797374656d2f61757468656e74696361
   74696f6e2f757365725b6e616d653d27626f62275d2f617574686f72697a
   65642d6b65795b6e616d653d2761646d696e275d2f6b65792d64617461
```

Third example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name='bob']"

CBOR encoding:

```
78 33
   2f696574662d73797374656d3a73797374656d2f61757468656e74696361
   74696f6e2f757365725b6e616d653d27626f62275d
```

7. Security Considerations

The security considerations of [RFC7049] and [RFC7950] apply.

This document defines an alternative encoding for data modeled in the YANG data modeling language. As such, this encoding does not contribute any new security issues in addition of those identified for the specific protocol or context for which it is used.

To minimize security risks, software on the receiving side SHOULD reject all messages that do not comply to the rules of this document and reply with an appropriate error message to the sender.

8. IANA Considerations

8.1. Tags Registry

This specification requires the assignment of CBOR tags for the following YANG datatypes. These tags are added to the Tags Registry as defined in section 7.2 of [RFC7049].

Tag	Data Item	Semantics	Reference
40	bits	YANG bits datatype	RFC XXXX
41	enumeration	YANG enumeration datatype	RFC XXXX
42	identityref	YANG identityref datatype	RFC XXXX
43	instance-identifier	YANG instance-identifier datatype	RFC XXXX

// RFC Ed.: update Tag values using allocated tags if needed and remove this note // RFC Ed.: replace XXXX with RFC number and remove this note

9. Acknowledgments

This document has been largely inspired by the extensive works done by Andy Bierman and Peter van der Stok on [I-D.ietf-core-comi]. [RFC7951] has also been a critical input to this work. The authors would like to thank the authors and contributors to these two drafts.

The authors would also like to acknowledge the review, feedback, and comments from Ladislav Lhotka and Juergen Schoenwaelder.

10. References

10.1. Normative References

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

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

10.2. Informative References

[I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-02 (work in progress), December 2017.

[I-D.ietf-core-sid] Veillette, M. and A. Pelov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-03 (work in progress), December 2017.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

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

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <https://www.rfc-editor.org/info/rfc7277>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov (editor)
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: http://www.landisgyr.com/

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

A. Keranen
Ericsson
M. Mohajer
u-blox UK

March 5, 2018

Media Types for FETCH & PATCH with Sensor Measurement Lists (SenML)
draft-keranen-core-senml-fetch-00

Abstract

The Sensor Measurement Lists (SenML) media type and data model can be used to send collections of resources, such as batches of sensor data or configuration parameters. The CoAP PATCH and FETCH methods enable accessing and updating parts of a resource or multiple resources with one request. This document defines two new media types that can be used with the CoAP PATCH and FETCH methods for resources represented with the SenML data model.

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 6, 2018.

Table of Contents

1. Introduction 2
2. Terminology 3
3. Media Type Definitions 3
3.1. SenML FETCH Media Type 3
3.1.1. Wild Card Selectors 4
3.2. SenML PATCH Media Type 5
4. Security Considerations 5
5. IANA Considerations 6
5.1. CoAP Content-Format Registration 6
5.2. senml-fetch+json Media Type 6
5.3. senml-patch+json Media Type 7
6. Acknowledgements 7
7. References 7
7.1. Normative References 7
7.2. Informative References 8
Authors' Addresses 8

1. Introduction

The Sensor Measurement Lists (SenML) media type [I-D.ietf-core-senml] and data model can be used to transmit collections of resources, such as batches of sensor data or configuration parameters.

Example of a SenML collection is shown below:

```
[
 {"bn":"2001:db8::2/3306/0/", "n":"5850", "vb":true},
 {"n":"5851", "v":42},
 {"n":"5750", "vs":"Ceiling light"}
]
```

Here three resources "3306/0/5850", "3306/0/5851", and "3306/0/5750", of an IPSO dimmable light smart object [IPSO] are represented using a single SenML Pack with three SenML Records. All resources share the same base name "2001:db8::2/3306/0/", hence full names for resources are "2001:db8::2/3306/0/5850", etc.

The CoAP [RFC7252] PATCH and FETCH methods [RFC8132] enable accessing and updating parts of a resource or multiple resources with one request.

This document defines two new media types that can be used with the CoAP PATCH and FETCH methods with resources represented with the SenML data model.

2. Terminology

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

Readers should also be familiar with the terms and concepts discussed in [RFC8132] and [I-D.ietf-core-senml]. Also the following terms are used in this document:

FETCH Record: One set of parameters that is used to match SenML Record(s). Presented using the SenML data model with extensions defined in this document.

FETCH Pack: One or more FETCH Records in an array structure.

PATCH Record: One set of parameters similar to FETCH Record but contains also instructions on how to change existing SenML Pack(s).

PATCH Pack: One or more PATCH Records in an array structure.

3. Media Type Definitions

The FETCH and PATCH media types for SenML are both modeled as extensions to the SenML media type to enable re-use of existing SenML parsers and generators, in particular on constrained devices. Unless mentioned otherwise, FETCH and PATCH Packs are constructed with the same rules and constraints as SenML Packs.

3.1. SenML FETCH Media Type

The FETCH media type is used to select and return parts of one or more SenML Packs. The SenML records are selected by giving the name(s) of the resources using the SenML "name" and/or "base name" Fields. The same rules for concatenating the name and base name Fields apply as for SenML Packs.

For example, to select resources "5850" and "5851" from the example in Section 1, the following FETCH Pack can be used:

```
[
 {"bn":"2001:db8::2/3306/0/", "n":"5850"},
 {"n":"5851"}
]
```

The result to a FETCH request with the example above would be:

```
[
 {"bn":"2001:db8::2/3306/0/", "n":"5850", "vb":true},
 {"n":"5851", "v":42},
]
```

When SenML records contain also time values, a name may no longer uniquely identify a single record. When no time is given in a FETCH Record, all SenML Records with the given name are be matched. When time is given in the FETCH Record, only a SenML Record (if any) with equal time value is matched.

3.1.1. Wild Card Selectors

Multiple SenML Records can be chosen simply by listing all names, and potentially times, in separate FETCH Records. However, for efficiency, also wild card operations for selecting multiple SenML records with a single FETCH Record MAY be supported.

Implementations that do not support wild card operations MUST return 4.00 "Bad Request" when receiving a request with wild cards.

TBD: better response code than 400?

The format for SenML wild card operations is for further study. This section proposes one possible format for discussion.

SenML names are often split to multiple segments using the "/" and/or ":" characters. For example, "3306/0/5850" contains three segments: "3306", "0", and "5850".

A new SenML Field, FETCH filter ("ff"), is defined for use with wild card operations. The filter expression is concatenated to the base name like the name field. The filter can contain asterisk ("*") characters that are used to match any set of characters within a SenML name until the next segment separator ("/" or ":").

TBD: more formal specification of matching rules

For example, the filter expression "3306/0/*" together with the base name "2001:db8::2/" would match all three Records in the example SenML Pack from Section 1.

Note that the "*" character is not a valid character in SenML names.

The "n" and "ff" Fields MUST NOT be used together.

Example of using the filter expression "3306/0/*" in a FETCH Pack is shown below:

```
[
 {"bn":"2001:db8::2/", "ff":"3306/0/*"}
]
```

Similarly, when used with the same base name, the filter expression "*/0/*" would match all the three Records and the filter expression "*/0/5850" would match only the Record with name "5850".

3.2. SenML PATCH Media Type

The PATCH media type is used to change the values of SenML Records. The names and times of the Records are given in same way as for the FETCH media type but PATCH Packs can include also new values and other SenML Fields for the Records.

For example, the following document could be given as PATCH payload to change/set values of two SenML Records for the example in Section 1:

```
[
 {"bn":"2001:db8::2/3306/0/", "n":"5850", "vb":false},
 {"n":"5851", "v":10}
]
```

If the request is successful, the resulting representation of the example SenML Pack would be as follows:

```
[
 {"bn":"2001:db8::2/3306/0/", "n":"5850", "vb":false},
 {"n":"5851", "v":10},
 {"n":"5750", "vs":"Ceiling light"}
]
```

4. Security Considerations

TBD

5. IANA Considerations

This document registers two new media types, one for the FETCH and one for the PATCH use, and CoAP Content-Format IDs for both media types.

5.1. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the SenML PATCH and FETCH media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. All IDs are assigned from the "Expert Review" (0-255) range. The assigned IDs are show in Table 1.

Media type	ID
application/senml-patch+json	TBD
application/senml-fetch+json	TBD

Table 1: CoAP Content-Format IDs

5.2. senml-fetch+json Media Type

Type name: application

Subtype name: senml-fetch+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. See RFC-AAAA for details. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: See Section 4 of RFC-AAAA.

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: Applications that use the SenML media type for resource representation.

Fragment identifier considerations: TBD

Additional information:

Magic number(s): none

File extension(s): senml-fetch-json

Windows Clipboard Name: "SenML fetch format"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-fetch-json conforms to public.text

Person & email address to contact for further information: Ari Keranen ari.keranen@ericsson.com

Intended usage: COMMON

Restrictions on usage: None

Author: Ari Keranen ari.keranen@ericsson.com

Change controller: IESG

5.3. senml-patch+json Media Type

TBD (similar to senml-fetch)

6. Acknowledgements

TBD

7. References

7.1. Normative References

[I-D.ietf-core-senml]
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-13 (work in progress), March 2018.

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

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

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

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

7.2. Informative References

[IPSO] IPSO, "IP for Smart Objects - IPSO Objects", 2018, <https://github.com/IPSO-Alliance/pub>.

Authors' Addresses

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Mojan Mohajer
u-blox UK

Email: Mojan.Mohajer@u-blox.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

A. Keranen
Ericsson

March 5, 2018

Too Many Requests Response Code for the Constrained Application Protocol
draft-keranen-core-too-many-reqs-00

Abstract

A Constrained Application Protocol (CoAP) server can experience temporary overload because one or more clients are sending requests to the server at a higher rate than the server is capable or willing to handle. This document defines a new CoAP Response Code for a server to indicate that a client should reduce the rate of requests.

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 6, 2018.

Table of Contents

1. Introduction 2
2. Terminology 2
3. CoAP Server Behavior 2
4. CoAP Client Behavior 3
5. Security Considerations 3
6. IANA Considerations 3
7. Acknowledgements 4
8. References 4
8.1. Normative References 4
8.2. Informative References 4
Author's Address 4

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] Response Codes are used by a CoAP server to indicate the result of the attempt to understand and satisfy a request sent by a client.

CoAP Response Codes are similar to the HTTP [RFC7230] Status Codes and many codes are shared with similar semantics by both CoAP and HTTP. HTTP has the code "429" registered for "Too Many Requests" [RFC6585]. This document registers a CoAP Response Code "4.29" for similar purpose and also defines use of the Max-Age option to indicate when a client can try the request again.

The 4.29 code is similar to the 5.03 "Service Unavailable" [RFC7252] code in a way that the 5.03 code can also be used by a server to signal an overload situation. However the 4.29 code indicates that the too frequent requests from the requesting client are the reason for the overload.

2. Terminology

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

Readers should also be familiar with the terms and concepts discussed in [RFC7252].

3. CoAP Server Behavior

If a CoAP server is unable to serve a client that is sending CoAP request messages more often than the server is capable or willing to handle, the server SHOULD respond to the request(s) with the Response Code 4.29, "Too Many Requests". The server MAY include in the

response a Max-Age option indicating the number of seconds after which the server assumes it is OK for the client to retry the request.

TBD: If a server chooses to (not)accept a request based on the method/resource, how should this be indicated in the reply?

4. CoAP Client Behavior

If a client receives the 4.29 Response Code from a CoAP server to a request, it SHOULD NOT send the same request to the server before the time indicated in the Max-Age option has passed. If the response does not contain Max-Age option, the client SHOULD wait for the Max-Age default value, 60 seconds.

A client MUST NOT rely on a server being able to send the 4.29 Response Code in an overload situation because an overloaded server may not be able to reply to all requests at all.

TBD: what kind of requests are (not) OK during the Max-Age? For example: the client MAY send a different request, in particular if the expected load for the server is smaller with that request?

5. Security Considerations

Replying to CoAP requests with a Response Code consumes resources from a server. For a server under attack it may be more appropriate to simply drop requests without responding.

If a CoAP reply with the Too Many Requests Response Code is not authenticated and integrity protected, an attacker can attempt to spoof a reply and make the client wait for an extended period of time before trying again.

6. IANA Considerations

IANA is requested to register the following Response Code in the "CoRE Parameters Registry", "CoAP Response Codes" sub-registry:

- Response Code: 4.29
- Description: Too Many Requests
- Reference: [[This document]]

7. Acknowledgements

This Response Code definition was originally part of the "Publish-Subscribe Broker for CoAP" document [I-D.ietf-core-coap-pubsub]. Author would like to thank Gyorgy Rethy and Sandor Katona for their contributions and reviews.

8. References

8.1. Normative References

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

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

8.2. Informative References

[I-D.ietf-core-coap-pubsub] Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-03 (work in progress), February 2018.

[RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <https://www.rfc-editor.org/info/rfc6585>.

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

Author's Address

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

B. Silverajan
Tampere University of Technology
T. Savolainen
Nokia Technologies
March 5, 2018

CoAP Communication with Alternative Transports
draft-silverajan-core-coap-alternative-transports-11

Abstract

The aim of this document is to provide a way forward to best decide upon how alternative transport information can be expressed in a CoAP URI. This draft examines the requirements for a new URI format for representing CoAP resources over alternative transports. Various potential URI formats are presented. Benefits and drawbacks of embedding alternative transport information in various ways within the URI components are also discussed. From all listed formats, the document finds scheme-based model to be the most technically feasible.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 6, 2018.

Table of Contents

1. Introduction 2
2. Conformance and Design Considerations 4
3. Situating Transport Information in CoAP URIs 5
3.1. Using the URI scheme component 5
3.1.1. Analysis 6
3.2. Using the URI authority component 6
3.2.1. Analysis 7
3.3. Using the URI path component 7
3.3.1. Analysis 7
3.4. Using the URI query component 7
3.4.1. Analysis 8
4. Discussion 8
5. IANA Considerations 8
6. Security Considerations 9
7. Acknowledgements 9
8. References 9
8.1. Normative References 9
8.2. Informative References 9
Appendix A. Expressing transport in the URI in other ways . . . 10
A.1. Transport information as part of the URI authority . . . 10
A.1.1. Usage of DNS records 11
A.2. Making CoAP Resources Available over Multiple Transports 12
A.3. Transport as part of a 'service:' URL scheme 13
Authors' Addresses 14

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a lightweight, binary application layer protocol designed for constrained environments. Owing to its operating environment, CoAP uses UDP and DTLS as its underlying transports between communicating endpoints. However, with an increase in deployment experiences as well as its popularity, compelling reasons exist for extending CoAP messaging to work over alternative transports. These allow CoAP to better address firewall and NAT traversal issues, to operate in Web browser-based and HTML5 applications as well as for energy-constrained M2M communication in cellular networks. At the time of writing, these transports are:

- TCP, TLS and Websockets [RFC8323]

- SMS for cellular networks[I-D.becker-core-coap-sms-gprs]

- SLIP for serial interfaces[I-D.bormann-t2trg-slipmux]

CoAP uses a REST-based model similar to HTTP, where URIs are used to identify resources at servers. An important factor of allowing CoAP communication over alternative transports, is to express not only the resource identifier, but also the alternative transport information in the URI.

CoAP URIs contain information, such as the endpoint address as well as the location of the resource hosted at the endpoint. CoAP URIs beginning with "coap://" are using UDP, while those beginning with "coaps://" are using DTLS.

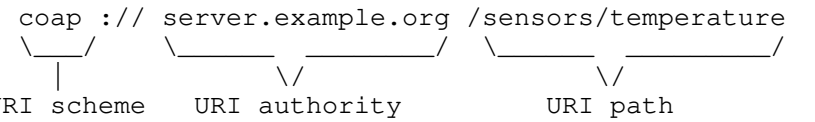


Figure 1: A CoAP URI

Figure 1 shows the structure of a simple example CoAP URI, in which the various URI components can beinterpreted as follows:

- The URI scheme component (e.g. "coap") contains an application-level identifier which typically identifies the protocol being used as well as its transport and network level protocol configurations. Such configurations are defined by convention or standardisation of the protocol using the scheme.

- The URI authority component ("server.example.com") contains the endpoint identification, which is typically a fully qualified domain name or a network-level host address.

- The URI path component ("/sensors/temperature") contains a parameterised resource identifier providing the location and identity of the resource at the endpoint.

In addition to these URI components, Figure 2 shows how specific queries on resource representations are provided by CoAP clients to servers, by specifying one or more URI query components in the URI.

```
coap :// server.example.org /sensors/temperature ?u=cel
                                                  \___/
                                                    |
                                                URI query
```

Figure 2: A CoAP URI with query

This document focuses on how CoAP URIs can be extended to contain information about alternative transports. For deriving the new URI format, the main design considerations are presented in the next section. Following that, various potential URIs are presented. These URIs provide examples of how transport identifiers can be situated in the URI scheme, authority, path or query components. The proposed URIs are analysed to select feasible formats while disqualifying those not meeting the design criteria.

2. Conformance and Design Considerations

In order to understand which URI formats are best suited for expressing transport information, certain considerations firstly need to be taken into account. Doing so eliminates URI formats that do not meet or conform to the stated requirements. The main criteria are:

1. Conformance to the generic syntax for a URI described in [RFC3986]. A URI format needs to be described in which each URI component clearly meets the syntax and percent-encoding rules described.

2. Alignment with best practices for URI design, as described in [RFC7320]. This is particularly important when it pertains to establishing or standardising the structure and usage of URIs with respect to the various URI components.

3. Request messages sent to a CoAP endpoint using a CoAP Transport URI may be responded to with a relative URI reference. [RFC3986] provides an algorithm to establish how relative references can be resolved against a base URI to obtain a target URI. Given this algorithm, a URI format needs to be described in which relative reference resolution does not result in a target URI that loses its transport-specific information

4. The URI can be supplied as a Proxy-Uri option by a CoAP end-point to a CoAP forward proxy. This allows communication with a CoAP end-point residing in a network using a different transport. Section 6.4 of [RFC7252] provides an algorithm for parsing a received URI to obtain the request's options. Conformance to

[RFC3986] is also necessary in order for the parsing algorithm to be successful.

In addition to the above mentioned requirements, where possible, the following considerations need to be borne in mind:

1. The URI format is able to represent a resource and the transport information for use in constrained environments, without requiring the presence of a naming infrastructure, such as DNS or a directory/lookup service.

2. Alternative transport information can be easily retrieved by computationally constrained nodes. In other words, the URI format does not result in unneccessarily complex code or logic in such nodes to parse and extract the transport to be used, nor the endpoint address.

3. URIs are designed to uniquely identify resources. When a single resource is represented with multiple URIs, URI aliasing [WWWArchv1] occurs. Avoiding URI aliasing is considered good practice.

4. CoAP URIs do not support fragment identifiers.

3. Situating Transport Information in CoAP URIs

The following subsections aim to describe potential URI formats in which the alternative transport information is placed in various URI components.

3.1. Using the URI scheme component

Expressing the transport information in the URI scheme component can be achieved by using new schemes. These can conform to an agreed-upon convention such as "coap+alternative_transport_name" for each new alternative transport and/or "coaps+alternative_transport_name" for its secure counterpart.

Examples of such URIs are:

- coap+tcp://server.example.org/sensors/temperature for using CoAP over TCP

- coap+sms://0015105550101/sensors/temperature for using CoAP over SMS with the endpoint identifier being a telephone subscriber number

- coaps+tcp://server.example.org/sensors/temperature for using CoAP over TLS

3.1.1. Analysis

Expressing transport information in the URI scheme delivers a URI which is human-readable and computationally as easy to parse as standard CoAP URIs, to extract transport identification information. The URI syntax conforms to [RFC3986], and relative URI resolution does not result in the loss of transport identification information. However, each new alternative transport requires minting new schemes, and IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [RFC7595]. Additionally, should a CoAP server wish to expose its resources over multiple transports (such as both UDP and TCP) , URI aliasing can occur if the URI scheme components of these multiple URIs differ in describing the same resource.

3.2. Using the URI authority component

Expressing the transport information within the authority component can result in two possible URI formats.

The first approach is to structure the URI authority's host sub-component with a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint_identifier".

Examples of resulting URIs are:

- coap://tcp-server.example.org/sensors/temperature for using CoAP over TCP

- coap://sms-0015105550101/sensors/temperature for using CoAP over SMS

The second approach is to hint at the alternative transport information, by explicitly specifying using the URI authority's port sub-component, thereby differentiating them from standard CoAP URIs.

Examples of resulting URIs are:

- coap://server.example.org:5684/sensors/temperature for using CoAP over TLS

- coap://server.example.org:80/sensors/temperature for using CoAP over WebSockets

3.2.1. Analysis

Embedding the transport information in the host would violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. Consequently, the host in a URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

Embedding the transport information in the port, on the other hand, would not violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. It would result in a CoAP URI that is less human-readable, but URI aliasing is minimised.

On the other hand, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form "//server2.example.org/path/to/another/resource", relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

3.3. Using the URI path component

Should the URI path component be used, then special characters or keywords need to be supplied in the path to make the transport explicit. Here, many proposals can exist. In general however, this will result in a URI format such as:

- coap://server.example.org/sensors/temperature;tcp for using CoAP over TCP, by appending the transport information at the end of the URI.

3.3.1. Analysis

Embedding the transport information in the URI path directly results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form "../../path/to/another/resource", relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI path component cannot be used as a basis for a new CoAP URI for alternative transports.

3.4. Using the URI query component

The alternative transport information, should URI query components be used, would result in a URI format such as:

- coap://server.example.org/sensors/temperature?alternative-transport=wss for using CoAP over secure WebSockets.

3.4.1. Analysis

Embedding the transport information in a URI query also results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form "../../path/to/another/resource", relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI query component cannot be used as a basis for a new CoAP URI for alternative transports.

4. Discussion

Based on the analysis of the various options for embedding alternative transport information in a CoAP URI, the most technically feasible option is to use the URI scheme component, as described in Section 3.1. To date, this has also been the WG consensus.

A discussion with IESG members during review of [RFC8323] revealed however, that using the URI scheme to express transport information is not desirable, to avoid the proliferation of new URI schemes for the same application-layer protocol. A strategy was instead proposed to preserve the existing CoAP URI and reuse it for alternative transports, by employing a combination of UDP Confirmable messages and timeouts to determine the eventual correct transport to use between a client and server [IESG-feedback]. The undertaken strategy would have obvious implications regarding interoperability, application and protocol logic, resource usage, for both new CoAP and existing CoAP implementations and deployments. Although URI aliasing can theoretically be avoided with this approach, at the time of writing, its technical feasibility over using the simpler strategy of using URI schemes, has yet to be validated. An obvious drawback is therefore that implementers and other SDOs may choose to provisionally or permanently register new URI schemes with IANA, for CoAP over alternative transports anyway, as was done by the Open Connectivity Foundation (OCF) [CoAP-TCP-TLS-registration].

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

New security risks are not envisaged to arise from the guidelines given in this document, for describing a new URI format containing transport identification within the URI scheme component. However, when specific alternative transports are selected for implementing support for carrying CoAP messages, risk factors or vulnerabilities can be present. Examples include privacy trade-offs when MAC addresses or phone numbers are supplied as URI authority components, or if specific URI path components employed for security-specific interpretations are accidentally encountered as false positives. While this document does not make it mandatory to introduce a security mode with each transport, it recommends ascribing meaning to the use of "coap+" and "coaps+" prefixes in the scheme component, with the "coaps+" prefix used for secure transports for CoAP messages.

7. Acknowledgements

Email discussions, comments and ideas from Thomas Fossati, Akbar Rahman, Klaus Hartke, Martin Thomson, Mark Nottingham, Dave Thaler, Graham Klyne, Carsten Bormann and Markus Becker greatly helped previous versions of this draft.

8. References

8.1. Normative References

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

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

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

8.2. Informative References

[CoAP-TCP-TLS-registration] , <https://www.iana>.

[I-D.becker-core-coap-sms-gprs]
Kuladinithi, K., Becker, M., Li, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-06 (work in progress), February 2017.

[I-D.bormann-t2trg-slipmux]
Bormann, C. and T. Kaupat, "Slipmux: Using an UART interface for diagnostics, configuration, and packet transfer", draft-bormann-t2trg-slipmux-02 (work in progress), January 2018.

[IESG-feedback]
, <https://www.ietf.org/mail-archive/web/core/current/msg08768>.

[RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates and Service: Schemes", RFC 2609, DOI 10.17487/RFC2609, June 1999, <https://www.rfc-editor.org/info/rfc2609>.

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

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

[WWWArchv1]
, <http://www.w3>.

Appendix A. Expressing transport in the URI in other ways

Other means of indicating the transport as a distinguishable component within the CoAP URI are possible, but have been deemed unsuitable by not meeting the design considerations listed, or are incompatible with existing practices outlined in [RFC7252]. They are however, retained in this section for historical documentation and completeness.

A.1. Transport information as part of the URI authority

A single URI scheme, "coap-at" can be introduced, as part of an absolute URI which expresses the transport information within the authority component. One approach is to structure the component with

a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint_identifier".

Examples of resulting URIs are:

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

- coap-at://sms-0015105550101/sensors/temperature

An implementation note here is that some generic URI parsers will fail when encountering a URI such as "coap-at://tcp-[2001:db8::1]/sensors/temperature". Consequently, an equivalent, but parseable URI from the ip6.arpa domain needs to be formulated instead. For [2001:db8::1] using TCP, this would result in the following URL:

coap-at://tcp-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa:5683/sensors/temperature

Usage of an IPv4-mapped IPv6 address such as [::ffff.192.100.0.1] can similarly be expressed with a URI from the ip6.arpa domain.

This URI format allows the usage of a single scheme to represent multiple types of transport end-points. Consequently, it requires consistency in ensuring how various transport-specific endpoints are identified, as a single URI format is used. Attention must be paid towards the syntax rules and encoding for the URI host component. Additionally, against a base URI of the form "coap-at://tcp-server.example.com/sensors/temperature", resolving a relative reference, such as "//example.net/sensors/temperature" would result in the target URI "coap-at://example.net/sensors/temperature", in which transport information is lost.

A.1.1. Usage of DNS records

DNS names can be used instead of IPv6 address literals to mitigate lengthy URLs referring to the ip6.arpa domain, if usage of DNS is possible.

DNS SRV records can also be employed to formulate a URL such as:

coap-at://srv-_coap._tcp.example.com/sensors/temperature

in which the "srv" prefix is used to indicate that a DNS SRV lookup should be used for _coap._tcp.example.com, where usage of CoAP over TCP is specified for example.com, and is eventually resolved to a numerical IPv4 or IPv6 address.

A.2. Making CoAP Resources Available over Multiple Transports

The CoAP URI used thus far is as follows:

```
URI         = scheme ":" hier-part [ "?" query ]
hier-part   = "//" authority path-abempty
```

A new URI format could be introduced, that does not possess an "authority" component, and instead defining "hier-part" to instead use another component, "path-rootless", as specified by RFC3986 [RFC3986]. The partial ABNF format of this URI would then be:

```
URI         = scheme ":" hier-part [ "?" query ]
hier-part   = path-rootless
path-rootless = segment-nz *( "/" segment )
```

The full syntax of "path-rootless" is described in [RFC3986]. A generic URI defined this way would conform to the syntax of [RFC3986], while the path component can be treated as an opaque string to indicate transport types, endpoints as well as paths to CoAP resources. A single scheme can similarly be used.

A constrained node that is capable of communicating over several types of transports (such as UDP, TCP and SMS) would be able to convey a single CoAP resource over multiple transports. This is also beneficial for nodes performing caching and proxying from one type of transport to another.

Requesting and retrieving the same CoAP resource representation over multiple transports could be rendered possible by prefixing the transport type and endpoint identifier information to the CoAP URI. This would result in the following example representation:

```
coap-at:tcp://example.com?coap://example.com/sensors/temperature
        \_______  ______/ \_________________  ___________________/
                \/                         \/
         Transport-specific            CoAP Resource
             Prefix
```

Figure 3: Prefixing a CoAP URI with TCP transport

Such a representation would result in the URI being decomposed into its constituent components, with the CoAP resource residing within the query component as follows:

Scheme: coap-at

Path: tcp://example.com

Query: coap://example.com/sensors/temperature

The same CoAP resource, if requested over a WebSocket transport, would result the following URI:

```
coap-at:ws://example.com/endpoint?coap://example.com/sensors/temperature
        \___________  __________/ \_________________  ____________________/
                    \/                              \/
            Transport-specific                 CoAP Resource
                  Prefix
```

Figure 4: Prefixing a CoAP URI with WebSocket transport

While the transport prefix changes, the CoAP resource representation remains the same in the query component:

Scheme: coap-at

Path: ws://example.com/endpoint

Query: coap://example.com/sensors/temperature

The URI format described here overcomes URI aliasing [WWWArchv1] when multiple transports are used, by ensuring each CoAP resource representation remains the same, but is prefixed with different transports. However, against a base URI of this format, resolving relative references of the form "//example.net/sensors/temperature" and "/sensor2/temperature" would again result in target URIs which lose transport-specific information.

Implementation note: While square brackets are disallowed within the path component, the '[' and ']' characters needed to enclose a literal IPv6 address can be percent-encoded into their respective equivalents. The ':' character does not need to be percent-encoded. This results in a significantly simpler URI string compared to section 2.2, particularly for compressed IPv6 addresses. Additionally, the URI format can be used to specify other similar address families and formats, such as Bluetooth addresses.

A.3. Transport as part of a 'service:' URL scheme

The "service:" URL scheme name was introduced in [RFC2609] and forms the basis of service description used primarily by the Service Location Protocol. An abstract service type URI would have the form "service:<abstract-type>:<concrete-type>"

where <abstract-type> refers to a service type name that can be associated with a variety of protocols, while the <concrete-type>

then providing the specific details of the protocol used, authority and other URI components.

Adopting the "service:" URL scheme to describe CoAP usage over alternative transports would be rather trivial. To use a previous example, a CoAP service to discover a Resource Directory and its base RD resource using TCP would take the form

service:coap:tcp://host.example.com/.well-known/core?rt=core-rd

The syntax of the "service:" URL scheme differs from the generic URI syntax and therefore such a representation should be treated as an opaque URI as Section 2.1 of [RFC2609] recommends.

Authors' Addresses

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

Teemu Savolainen
Nokia Technologies
Hatanpaeaen valtatie 30
FI-33100 Tampere
Finland

Email: teemu.savolainen@nokia.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

B. Silverajan
TUT
M. Ocak
Ericsson
March 5, 2018

CoAP Protocol Negotiation
draft-silverajan-core-coap-protocol-negotiation-08

Abstract

CoAP has been standardised as an application-level REST-based protocol. When multiple transport protocols exist for exchanging CoAP resource representations, this document introduces a way forward for CoAP endpoints as well as intermediaries to agree upon alternate transport and protocol configurations as well as URIs for CoAP messaging. Several mechanisms are proposed: Extending the CoRE Resource Directory with new parameter types, introducing a new CoAP Option with which clients can interact directly with servers without needing the Resource Directory, and finally a new CoRE Link Attribute allowing exposing alternate locations on a per-resource basis.

Table of Contents

1. Introduction 2
2. Aim 4
2.1. Overcoming Middlebox Issues 4
2.2. Better resource caching and serving in proxies 5
2.3. Interaction with Energy-constrained Servers 6
3. Node Types based on Transport Availability 7
4. New Resource Directory Parameters 8
4.1. The 'at' RD parameter 8
4.2. The 'tt' RD parameter 9
5. CoAP Alternative-Transport Option 11
6. The 'ol' CoRE Link Attribute 14
6.1. Using /.well-known/core 14
6.2. Using CoRE Resource Directory 15
7. IANA Considerations 15
8. Security Considerations 15
9. Acknowledgements 16
10. References 16
10.1. Normative References 16
10.2. Informative References 16
Appendix A. Change Log 17
A.1. From -07 to -08 17
A.2. From -06 to -07 17
A.3. From -05 to -06 17
A.4. From -04 to -05 17
A.5. From -03 to -04 17
A.6. From -02 to -03 17
A.7. From -01 to -02 18
A.8. From -00 to -01 18
Authors' Addresses 18

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] allows clients, origin servers and proxies, to exchange and manipulate resource representations using REST-based methods over UDP or DTLS. CoAP messaging however can use other alternative underlying transports [I-D.silverajan-core-coap-alternative-transports].

When CoAP-based endpoints and proxies possess the ability to perform CoAP messaging over multiple transports, significant benefits can be obtained if communicating client endpoints can discover that multiple transport bindings may exist on an origin server over which CoAP resources can be retrieved. This allows a client to understand and possibly substitute a different transport protocol configuration for the same CoAP resources on the origin server, based on the preferences of the communicating peers. Inevitably, if two CoAP endpoints reside in distinctly separate networks with orthogonal transports, a CoAP proxy node is needed between the two networks so that CoAP Requests and Responses can be exchanged properly.

A URI in CoAP, however, serves two purposes simultaneously. It firstly functions as a locator, by specifying the network location of the endpoint hosting the resource, and the underlying transport used by CoAP for accessing the resource representation. It secondly identifies the name of the specific resource found at that endpoint together with its namespace, or resource path. A single CoAP URI cannot be used to express the identity of the resource independently of alternate underlying transports or protocol configuration. Multiple URIs can result for a single CoAP resource representations if:

- the authority components of the URI differ, owing to the same physical host exposing several network endpoints. For example, "coap://example.org/sensors/temperature" and "coap://example.net/sensors/temperature"

- the scheme components of the URI differ, owing to the origin server exposing several underlying transport alternatives. For example, "coap://example.org/sensors/temperature" and "coap+tcp://example.org/sensors/temperature"

Without a priori knowledge, clients would be unable to ascertain if two or more URIs provided by an origin server are associated to the same representation or not. Consequently, a communication mechanism needs to be conceived to allow an origin server to properly capture the relationship between these alternate representations or locations and then subsequently supply this information to clients. This also goes some way in limiting URI aliasing [WWWArchv1].

In order to support CoAP clients, proxies and servers wishing to use CoAP over multiple transports, this draft proposes the following:

- An ability for servers to register supported CoAP transports to a CoRE Resource Directory [I-D.ietf-core-resource-directory] with optional registration lifetime values

- A means for CoAP clients to interact with a CoRE resource directory interface for requesting and discovering alternative transports and locations of CoAP resources

- New Resource Directory parameter types enabling the above-mentioned features.

- A new CoAP Option called Alternative-Transport that can be used by CoAP clients to discover and retrieve the types of alternative transports available at the origin server, as well as the links describing the transport-specific endpoint address at which CoAP resources are exposed from.

- A new CoRE Link attribute for exposing transports and endpoint locations on an origin server on a per-resource basis.

2. Aim

The following simple scenarios aim to better portray how CoAP protocol negotiation benefits communicating nodes

2.1. Overcoming Middlebox Issues

Discovering which transports are available is important for a client to determine the optimal alternative to perform CoAP messaging according to its needs, particularly when separated from a CoAP server via a NAT. It is well-known that some firewalls as well as many NATs, particularly home gateways, hinder the proper operation of UDP traffic. NAT bindings for UDP-based traffic do not have as long timeouts as TCP-based traffic.

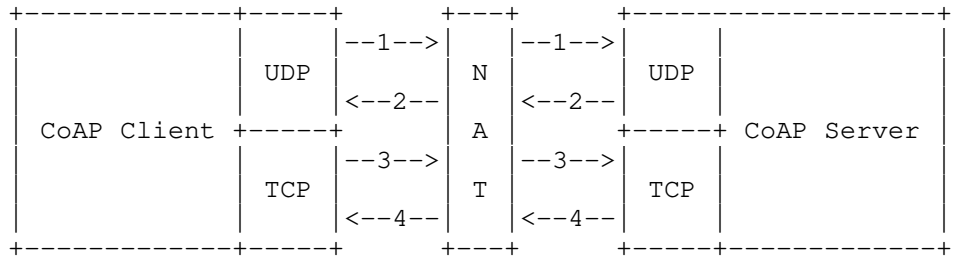


Figure 1: CoAP Client initially accesses CoAP Server over UDP and then switching to TCP

Figure 1 depicts such a scenario, where a CoAP client residing behind a NAT uses UDP initially for accessing a CoAP Server, and engages in discovering alternative transports offered by the server. The client subsequently decides to use TCP for CoAP messaging instead of UDP to set up an Observe relationship for a resource at the CoAP Server, in order to avoid incoming packets containing resource updates being discarded by the NAT.

2.2. Better resource caching and serving in proxies

Figure 2 outlines a more complex example of intermediate nodes such as CoAP-based proxies to intelligently cache and respond to CoAP or HTTP clients with the same resource representation requested over alternative transports or server endpoints. As with the earlier example, the CoAP Server registers its transports to a Resource Directory (This is assumed to be performed beforehand and not depicted in the figure, for brevity)

In this example, a CoAP over WebSockets client successfully obtains a response from a CoAP forward proxy to retrieve a resource representation from an origin server using UDP, by supplying the CoAP server's endpoint address and resource in a Proxy-URI option. Arrow 1 represents a GET request to "coap+ws://proxy.example.com" which subsequently retrieves the resource from the CoAP server using the URI "coap://example.org/sensors/temperature", shown as arrow 2.

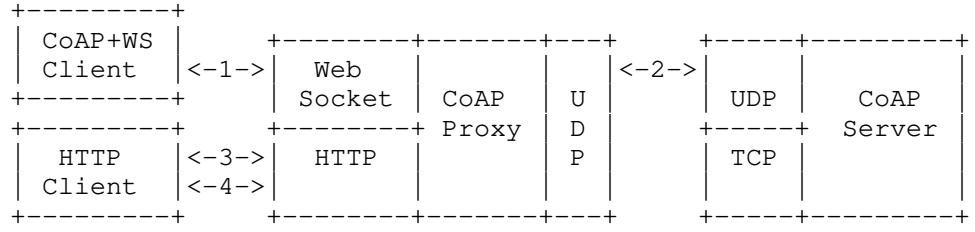


Figure 2: Proxying and returning a resource's alternate cached representations to multiple clients

Subsequently, assume an HTTP client requests the same resource, but instead specifies a CoAP over TCP alternative URI instead. Arrow 3 represents this event, where the HTTP client performs a GET request to "http://proxy.example.com/coap+tcp://example.org/sensors/temperature". When the proxy receives the request, instead of immediately retrieving the temperature resource again over TCP, it

first verifies either from the Resource Directory or directly from the server, whether the cached resource retrieved over UDP is a valid equivalent representation of the resource requested by the HTTP client over TCP. Upon confirmation, the proxy is able to supply the same cached representation to the HTTP client as well (arrow 4).

2.3. Interaction with Energy-constrained Servers

Figure 3 illustrates discovery and communication between a CoAP client and an energy-constrained CoAP Server. Such a server aims at conserving its energy unless a need arises otherwise. The figure first depicts the server registering itself to a Resource Directory over IP, and also supplies its alternative CoAP transport endpoints (in this case, SMS), in steps 1 and 2. The server can subsequently disable communication radio interfaces requiring greater energy (such as for IP-based comunication), powering it up sporadically for maintenance activities like registration renewals. At other times, it maintains communication in a low-power state by listening only for incoming SMS messages.

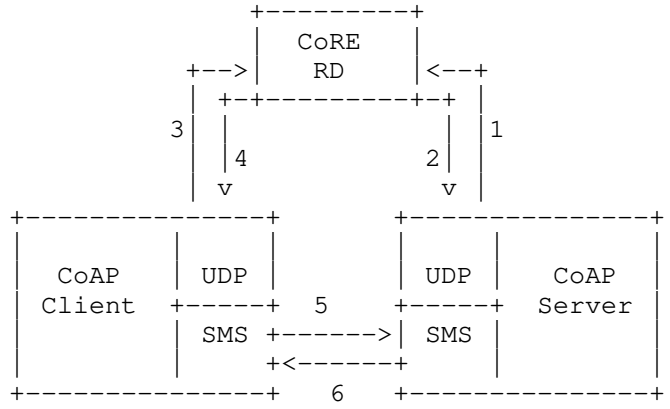


Figure 3: CoAP client interacting with RD to discover a server's SMS-based endpoint

A CoAP client wishing to perform CoAP operations with an energy-constrained CoAP server may query a resource directory for the SMS-based endpoint of the server (steps 3 and 4). Subsequently, SMS-based CoAP communication can occur between the endpoints as shown by arrows 5 and 6. Alternatively, the incoming SMS can be also used by the server as a triggering event to temporarily power up its radio

interface so that UDP or other transport-based CoAP communication can instead be employed for low latency communication with the client.

3. Node Types based on Transport Availability

In [RFC7228], Tables 1, 3 and 4 introduced classification schemes for devices, in terms of their resource constraints, energy limitations and communication power. For this document, in addition to these capabilities, it seems useful to also identify devices based on their transport capabilities.

Name	Transport Availability
T0	Single transport
T1	Multiple transports, with one or more active at any point in time
T2	Multiple active and persistent transports at all times

Table 1: Classes of Available Transports

Type T0 nodes possess the capability of exactly 1 type of transport channel for CoAP, at all times. These include both active and sleepy nodes, which may choose to perform duty cycling for power saving.

Type T1 nodes possess multiple different transports, and can retrieve or expose CoAP resources over any or all of these transports. However, not all transports are constantly active and certain transport channels and interfaces could be kept in a mostly-off state for energy-efficiency, such as when using CoAP over SMS.

Type T2 nodes possess more than 1 transport, and multiple transports are simultaneously active at all times in a persistent manner. CoAP proxy nodes which allow CoAP endpoints from disparate transports to communicate with each other, are a good example of this.

4. New Resource Directory Parameters

In order to allow resource interactions between clients and servers with multiple locations or transports, the registration, update and lookup interfaces of the CoRE Resource Directory need to be extended. In this section two new RD parameters, "at" and "tt" are introduced. Both are optional CoAP features. If supported, they occur at the granularity level of an origin server, ie. they cannot be applied selectively on some resources only. When absent, it is assumed that the server does not support multiple transports or locations.

4.1. The 'at' RD parameter

A CoAP server wishing to advertise its resources over multiple transports does so by using one or more "at" parameters to register CoAP alternative transport URIs with a Resource Directory. Such a URI would contain the scheme, address as well as any port or paths at which the server is available.

Name	Query	Validity	Description
CoAP Transport URI List	at	URI	URI (scheme, address, port and path) available at the server

Table 2: The "at" RD parameter

The "at" parameter extends the Resource Directory's Registration and Update interfaces.

The following example shows a type T1 endpoint registering its resources and advertising its ability to use TCP and WebSockets as alternative transports:

```
Req: POST coap://rd.example.com/rd?ep=node1
   &at=coap+tcp://[2001:db8:f1::2]&at=coap+ws://server.example.com
Content-Format: 40
Payload:
</temperature>;ct=0;rt="temperature";if="core.s"

Res: 2.01 Created
Location: /rd/1234
```

An endpoint lookup would just reflect the registered attributes:

```
Req: GET /rd-lookup/ep

Res: 2.05 Content
</rd/1234>;ep="node1";con="coap://[2001:db8:f1::2]:5683";
   at="coap+tcp://[2001:db8:f1::2]";at="coap+ws://server.example.com"
```

The next example shows the same endpoint updating its registration with a new lifetime and the availability of a single alternative transport for CoAP (in this case TCP):

```
 Req: POST /rd/1234?lt=600
    &at=coap+tcp://[2001:db8:f1::2]
 Content-Format: 40
 Payload:
 </temperature>;ct=0;rt="temperature";if="core.s"

 Res: 2.04 Changed
```

If a lookup is performed on the same endpoint only 1 alternative transport is indicated:

```
 Req: GET /rd-lookup/ep

 Res: 2.05 Content
 </rd/1234>;ep="node1";con="coap://[2001:db8:f1::2]:5683";
    at="coap+tcp://[2001:db8:f1::2]"
```

A UDP client would then see the following in a resource lookup:

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

Res: 2.05 Content
<coap://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";if="core.s";
   anchor="coap://[2001:db8:f1::2]"
```

4.2. The 'tt' RD parameter

A CoAP client wishing to perform a look-up on the Resource Directory for CoAP servers supporting multiple transports does so by using a new "tt" parameter to query for CoAP alternative transport URIs.

Name	Query	Validity	Description
CoAP Transport Type	tt		Transport type requested by the client

Table 3: The "tt" RD parameter

The "tt" parameter extends the Resource Directory's rd-lookup interface.

The following example shows a client performing a lookup for endpoints supporting TCP:

```
Req: GET /rd-lookup/ep?tt=tcp

Res: 2.05 Content
</rd/1234>;con="coap+tcp://[2001:db8:f1::2]";ep="node1";ct="40"
```

The following example shows a client performing a resource lookup for endpoints supporting TCP:

```
Req: GET /rd-lookup/res?rt=temperature&tt=tcp

Res: 2.05 Content
<coap+tcp://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";
 if="core.s";anchor="coap+tcp://[2001:db8:f1::2]"
```

The following example shows a client performing a lookup for endpoints supporting SMS i.e. discovering SMS transports for sleepy nodes and using SMS to communicate with the endpoint:

```
Req: GET /rd-lookup/ep?et=oic.d.switch&tt=sms

Res: 2.05 Content
</rd/2345>;con="coap+sms://0015105550101/";ep="node5";
 et="oic.d.switch";ct="40",
</rd/4521>;con="coap+sms://0015105550202/";ep="node8";
 et="oic.d.switch";ct="40"
```

5. CoAP Alternative-Transport Option

The CoAP Alternative-Transport Option can be used by CoAP clients and CoAP servers in both Request and Response messages in constrained environments where a CoRE Resource Directory is not present.

Figure 4 depicts the properties of the Alternative-Transport Option.

No.	C	U	N	R	Name	Format	Length	Default
66		x	-	x	Alternative-Transport	string	0-1034	(none)

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

Figure 4: The Alternative-Transport Option

When included in a Request message, this option is used by the client in 2 possible ways. In the first case, a CoAP client can include the Option with Length 0 to retrieve all alternative transports from a CoAP server. In response to the client, the server includes base URI for each transport in its own Option. In the second case, a CoAP client can include the Option with a specific value in a CoAP Request, and the CoAP server returns the base URI(s) for the specified transport. If the specified transport by a CoAP client returns multiple results on a CoAP server, the server returns all base URIs of the transport in the response, each base URI in its own Option.

A CoAP client can also use this Option to retrieve several transports at once by including multiple Options in the request to a CoAP server. If any of the specified transports is supported by the server, the server returns all base URIs in its own option. There can be more than 1 result for any of the transports so that each transport base URI is still included in the response in its own option.

Figure 5 describes a simple interaction between a client and a server, in which the client uses an Alternative-Transports Option with a null value to discover and retrieve all the available transports from the server, as part of a GET operation to retrieve a

resource representation. The server responds with a CoAP Response message which contains the resource representation as a payload. In addition, the server also supplies multiple Alternative-Transport Options in the message, with each Option containing the base URI for an available transport. In this case the base URIs returned for TCP-based and WebSocket transports indicate their availability over a non-standard port.

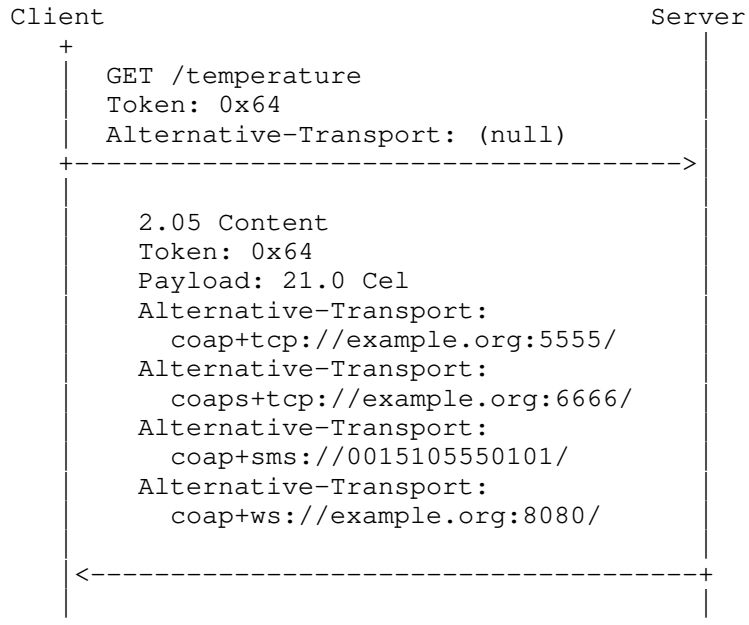


Figure 5: Requesting all available alternative transports on the server, and their locations

Alternatively, a client can also request for the availability of a specific transport on the server, as shown in Figure 6. Here, the CoAP Request contains an Alternative-Transport Option with the value set to request the Base URIs for TCP-based endpoints.

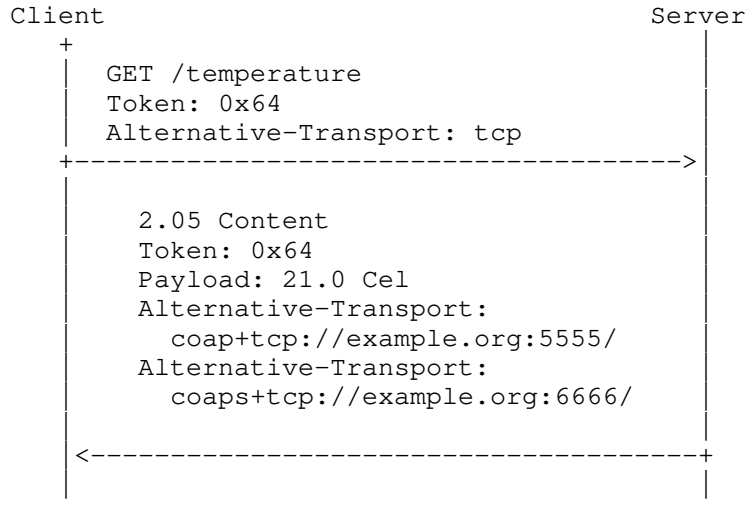


Figure 6: Requesting TCP-based alternative transports on the server, and their locations

A client may also request a subset of available transports on the server, by providing multiple Options, each having a single transport identifier. The server likewise responds to the client request by supplying the requested transport information. This is shown in Figure 7.

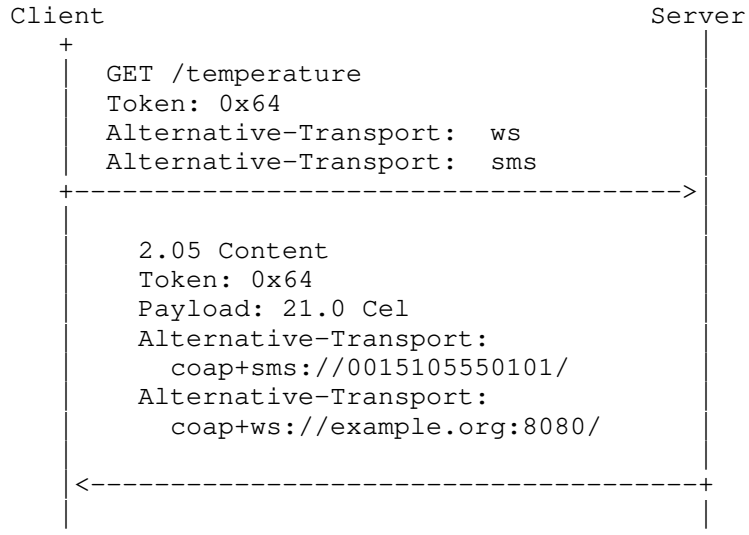


Figure 7: Requesting WebSocket- and SMS-based alternative transports on the server, and their locations

6. The 'ol' CoRE Link Attribute

In the majority of cases, it is expected that an origin server would expose all its resources uniformly on its available transports or endpoint addresses. Exceptions can exist however, where alternate locations are made available on a per-resource basis. For such cases, a new 'ol' ("other locations") attribute is provided. One or more 'ol' attributes are used to provide base URIs from which a specific resource can be reached. Allowing per-resource endpoint or transport availability enables specific functions such as firmware updates or hardware-specific operations. It also facilities mapping to and from OCF-based resource-specific endpoint descriptions.

6.1. Using /.well-known/core

REQ: GET /.well-known/core

RES: 2.05 Content
</sensors/temp>;ct=41;rt="temperature-f";if="sensor",
</sensors/door>;ct=41;rt="door";if="sensor",
</sensors/light>;if="sensor"; ol="http://[FDFD::123]:61616";
ol="coap://server2.example.com"

6.2. Using CoRE Resource Directory

```
   Req: POST coap:/rd.example.com/rd
    ?ep=node1&at=coap+tcp://server.example.com&at=coap+ws://server.example.com:5
683/ws/
   Content-Format: 40
   Payload:
   </sensors/temp>;ct=41;rt="temperature-f";if="sensor",
   </sensors/door>;ct=41;rt="door";if="sensor",
   </sensors/light>;if="sensor"; ol="http://[FDFD::123]:61616";
     ol="coap://server2.example.com"

   Res: 2.01 Created
   Location: /rd/4521
```

7. IANA Considerations

This document requests the registration of new RD parameter types "at" and "tt".

The following entry needs to be added to the CoAP Option Numbers Registry:

Number	Name	Reference
66	Alternative-Transports	(this document)

8. Security Considerations

When multiple transports, locations and representations are used, some obvious risks are present both at the origin server as well as by requesting clients.

When a client is presented with alternate URIs for retrieving resources, it presents an opportunity for attackers to mount a series of attacks, either by hijacking communication and masquerading as an alternate location or by using a man-in-the-middle attack on TLS-based communication to a server and redirecting traffic to an alternate location. A malicious or compromised server could also be used for reflective denial-of-service attacks on innocent third

parties. Moreover, clients may obtain web links to alternate URIs containing weaker security properties than the existing session.

9. Acknowledgements

Thanks to Jaime Jimenez, Christian Amsuess and Klaus Hartke for comments and reviewing this draft. Teemu Savolainen was involved in initial discussions about protocol negotations and lifetime values. Zach Shelby provided significant suggestions on how the Resource Directory can be employed and extended in place of link attributes and relation types.

10. References

10.1. Normative References

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

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

10.2. Informative References

[I-D.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-11 (work in progress), March 2018.

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

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

[WWWArchv1]
http://www.w3.org/TR/webarch/#uri-aliases, "Architecture of the World Wide Web, Volume One", December 2004.

Appendix A. Change Log

A.1. From -07 to -08

Added example of energy constrained CoAP server

Updated examples of using "at" and "tt"

"at" and "ol" are no longer comma-separated URI lists.

A.2. From -06 to -07

Added support for 'ol' Link attribute

A.3. From -05 to -06

Added support for CoAP Alternative-Transports Option

A.4. From -04 to -05

Freshness update

A.5. From -03 to -04

Removed previously introduced link attribute and relation types

Initial foray with Resource Directory support

A.6. From -02 to -03

Added new author

Rewrite of "Introduction" section

Added new Aims Section

Added new Section on Node Types

Introduced "al" Active Lifetime link attribute

Added new Section on Observing transports and resources

Security and IANA considerations sections populated

A.7. From -01 to -02

Freshness update.

A.8. From -00 to -01

Reworked "Introduction" section, added "Rationale", and "Goals" sections.

Authors' Addresses

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
FI-33720 Tampere
Finland

Email: bilhanan.silverajan@tut.fi

Mert Ocak
Ericsson
Hirsalantie 11
02420 Jorvas
Finland

Email: mert.ocak@ericsson.com

Network Working Group A. Minaburo
Internet-Draft Acklio
Intended status: Informational L. Toutain
Expires: April 29, 2018 Institut Mines Telecom Atlantique
October 26, 2017

CoAP Time Scale Option
draft-toutain-core-time-scale-00

Abstract

SCHC compression mechanism for LPWAN network enables IPv6 on devices connected to a constrained network (LPWAN). They can communicate with a CoAP server located anywhere in the Internet. LPWAN network characteristics limits the number of exchanges and may impose a long RTT. The CoAP server must be aware of these properties to manage correctly requests. The Time Scale option allows a device to inform a CoAP server of the duration the message ID value should be kept in memory to manage correctly message duplication.

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. CoAP Message ID

Constraint Application Protocol (CoAP) [RFC7252] implements a simple reliable transport mechanism based on ARQ. Each CoAP message contains a 16 bit Message ID (noted afterward MID). A client selects a MID in a CON message and expects an ACK message containing the same MID value. A timer makes the client resend the request if no ACK is received during a pre-defined period.

To avoid a second process of duplicated requests by the server, a list of messages ID already acknowledged must be maintained for a period of time. If the message ID is already in the list, the message is just acknowledged and not processed by upper layer. Therefore, the client cannot use this MID value in another request during the same period of time.

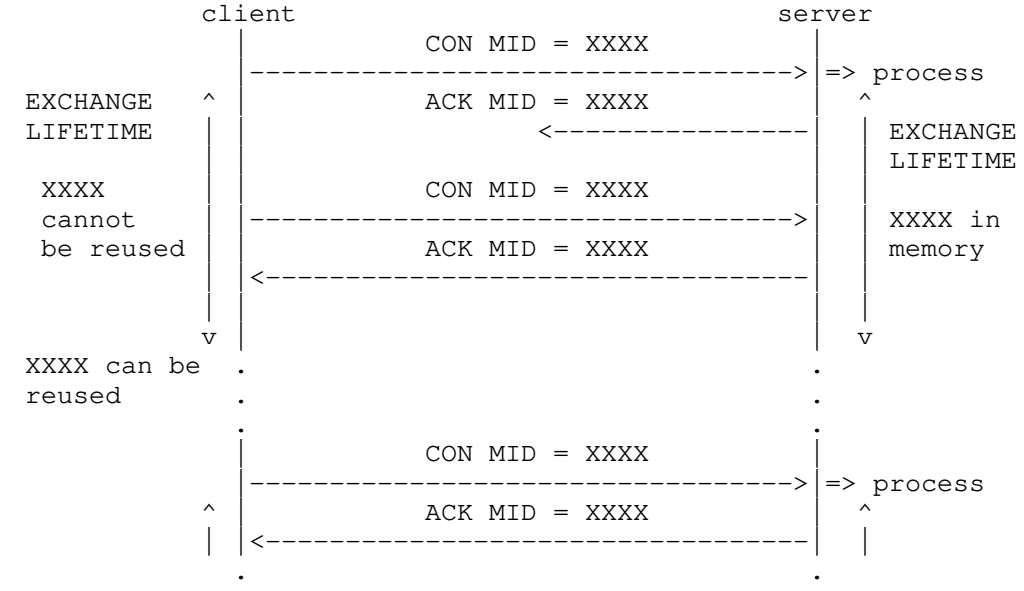


Figure 1: Delayed transmission.

[RFC7252] calls the period a MID is assigned to a request the EXCHANGE_LIFETIME. The value is based on the worst case scenario

taking into account the propagation time, the number of retransmissions and the processing time. The default value for EXCHANGE_LIFETIME is set to 247 seconds for MAX_RTT of 202 seconds.

2. LPWAN networks

Low Power Wide Area Network (LPWAN) family regroups networks dedicated to the Internet of Things. They provide a large coverage with a limited energy consumption. They mostly use the license-free ISM band. The [I-D.ietf-lpwan-overview] gives an overview of the technology and the star oriented topology architecture. A Network Gateway (NGW) is at the interconnection between the LPWAN and the Internet network.

To ensure fairness among nodes, regulation imposes a duty cycle. In practice, with a 1% duty cycle, a node sending a message of s seconds must wait 99 x s seconds before sending another message. For instance, in some technologies sending a 50 bytes message takes 2 seconds, forcing a silence of 198 seconds.

The device sleeps most of the time to preserve energy. If a device can use the uplink channel at any time, downlink channel is generally available during a short receiving window following the message emission. Therefore a message sent to a device out of this receiving window will be lost. Network Gateways are aware of this restriction and buffers downlink messages until an uplink message is received which opens the receiving window.

Figure 2 illustrates this. A CoAP client sends a request every hour. Even if the server replies immediately, the answer may be buffered by the Network GW until an new uplink message is sent. In that case, the client will only receive the answer after one hour when the next request is sent. The RTT is influenced by the message periodicity and the EXCHANGE_LIFTEIME value can be computed locally by client to dimension its timers.

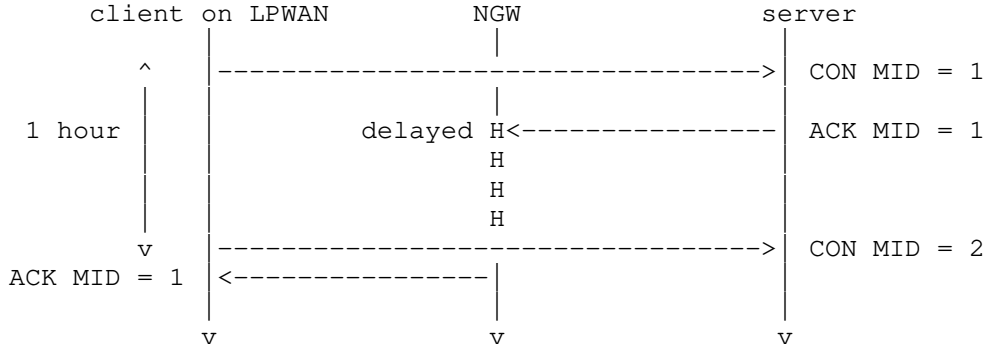


Figure 2: Delayed transmission.

The server should remain as generic as possible and EXCHANGE_LIFETIME parameter has to be adapted to the client behavior. If the period is too large, the server will have to memorize a longer list of MID for fast responding client. On the other hand, if the EXCHANGE_LIFETIME is too short, this leads to misbehaviors as shown in Figure 3, a retransmission will be viewed as a new request.

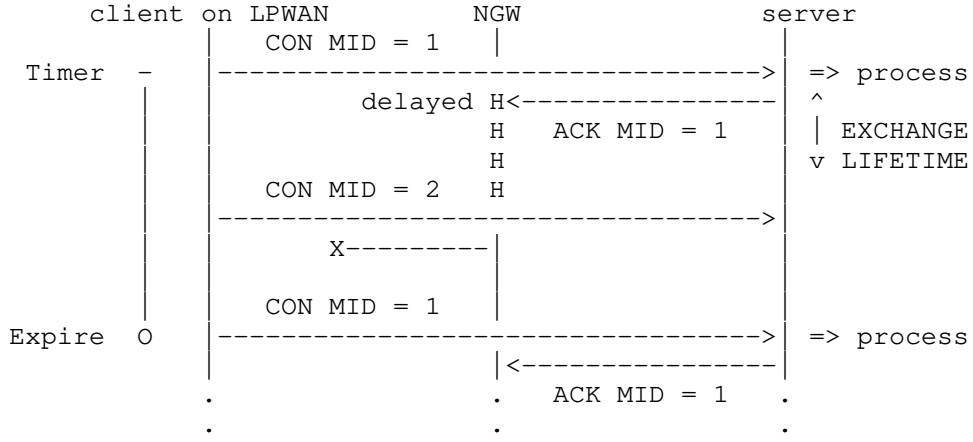


Figure 3: Retransmission.

The Time Scale option, added into all the CoAP requests, informs the server of the duration a message ID should be memorized into the server and therefore the duration during which a client should not reuse the same message ID for a new request. This way, the server can adapt its behavior to different environments.

It is important to notice that this option will not contribute to an DoS attack. This option does not increase the number of message ID memorized by the server. In fact, the Time Scale option can be viewed as a contract between the client and the server, which means that the client will send a reasonable number of request during that period. The number of memorized message ID is independent of the duration of the exchange but linked to the number a simultaneous request a client can send. If a client is sending a number of request larger than expected, they can be easily discarded by the server.

3. Timescale Option

Timescale is a new CoAP option that tells the server how many seconds the MID should be memorized by the server. This option must be included in all the exchanges coming from a high latency device.

Number	C	U	N	R	Name	Format	Length	Default
259	X				Time Scale	uint	1-4	3600

Figure 4: Time Scale Option.

This option is critical, if a server does not recognize it, it must inform the client that EXCHANGE_LIFETIME cannot be modified. The option is Safe-to-forward so a proxy does not have to understand this option, since only the server is concerned with the MID management. The value (in seconds) contains the new EXCHANGE_LIFETIME set by the server for this request. If the value is smaller than the default value, this option is discarded and the client receives an error message.

4. Normative References

[I-D.ietf-lpwan-overview]
Farrell, S., "LPWAN Overview", draft-ietf-lpwan-overview-07 (work in progress), October 2017.

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

Authors' Addresses

Ana Minaburo
Acklio
2bis rue de la Chataigneraie
35510 Cesson-Sevigne Cedex
France

Email: ana@ackl.io

Laurent Toutain
Institut Mines Telecom Atlantique
2 rue de la Chataigneraie
CS 17607
35576 Cesson-Sevigne Cedex
France

Email: Laurent.Toutain@imt-atlantique.fr

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: July 28, 2018

M. Veillette, Ed.
Trilliant Networks Inc.

January 24, 2018

Constrained YANG Module Library
draft-veillette-core-yang-library-02

Abstract

This document describes a YANG library that provides information about all the YANG modules used by a constrained network management server (e.g., a CoAP Management Interface (CoMI) server). Simple caching mechanisms are provided to allow clients to minimize retrieval of this information.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 28, 2018.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Major differences between ietf-constrained-yang-library and ietf-yang-library 3
2. Terminology and Notation 3
3. Overview 4
3.1. Tree diagram 4
3.2. Description 5
3.2.1. modules-state 5
3.2.2. modules-state/hash 5
3.2.3. modules-state/module 5
4. YANG Module "ietf-constrained-yang-library" 5
5. IANA Considerations 10
5.1. YANG Module Registry 10
6. Security Considerations 10
7. Acknowledgments 11
8. References 11
8.1. Normative References 11
8.2. Informative References 11
Author's Address 12

1. Introduction

WARNING: Both this contribution and the CoMI protocol [I-D.ietf-core-comi] need to be reviewed to verify their compatibility with the "Network Management Datastore Architecture" (NMDA). See [I-D.dsdt-nmda-guidelines], [I-D.ietf-netconf-rfc7895bis], [I-D.ietf-netmod-revised-datastores] and [I-D.ietf-netconf-nmda-restconf] for more details.

The YANG library specified in this document is available to clients of a given server to discover the YANG modules supported by this constrained network management server. A CoMI server provides a link to this library in the /mod.uri resource. The following YANG module information is provided to client applications to fully utilize the YANG data modeling language:

- module list: The list of YANG modules implemented by a server, each module is identified by its assigned YANG Schema Item iDentifier (SID) and revision.

- submodule list: The list of YANG submodules included by each module, each submodule is identified by its assigned SID and revision.

- feature list: The list of features supported by the server, each feature is identified by its assigned SID.

- deviation list: The list of YANG modules used for deviation statements associated with each YANG module, each module is identified by its assigned SID and revision.

1.1. Major differences between ietf-constrained-yang-library and ietf-yang-library

YANG module 'ietf-constrained-yang-library' targets the same functionality and shares the same approach as YANG module ietf-yang-library. The following changes with respect to ietf-yang-library are specified to make ietf-constrained-yang-library compatible with SID [I-D.ietf-core-yang-cbor] used by CoMI [I-D.ietf-core-comi] and to improve its applicability to constrained devices and networks.

- YANG module 'ietf-constrained-yang-library' extends the caching mechanism supported by 'ietf-yang-library' to multiple servers of the same type. This is accomplished by replacing 'module-set-id' by a hash of the library content.

- Modules, sub-modules, deviations and features are identified using a numerical value (SID) instead of a string (yang-identifier).

- The "namespace" leaf, not required for SIDs, but mandatory in 'ietf-yang-library' is not included in 'ietf-constrained-yang-library'.

- Schemas can be located using the already available module or sub-module identifier (SID) and revision. For this reason, support of module and sub-module schema URIs have been removed.

- To minimize their size, each revision date is encoded in binary.

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- module
- submodule
- feature
- deviation

The following terms are defined in [I-D.ietf-core-yang-cbor]:

- YANG Schema Item iDentifier (SID)

The following terms are defined in [I-D.ietf-core-comi]:

- client
- server

The following terms are used within this document:

- library: a collection of YANG modules used by a server.

3. Overview

The "ietf-constrained-yang-library" module provides information about the YANG library used by a given server. This module is defined using YANG version 1 as defined by [RFC7950], but it supports the description of YANG modules written in any revision of YANG.

3.1. Tree diagram

The tree diagram of YANG module ietf-constrained-yang-library is provided below. This graphical representation of a YANG module is defined in [I-D.ietf-netmod-yang-tree-diagrams].

```
module: ietf-constrained-yang-library
   +--ro modules-state
      +--ro hash      binary
      +--ro module* [sid revision]
         +--ro sid                 comi:sid
         +--ro revision            revision
         +--ro feature*            comi:sid
         +--ro deviation* [sid revision]
         |  +--ro sid          comi:sid
         |  +--ro revision     revision
         +--ro conformance-type    enumeration
         +--ro submodule* [sid revision]
            +--ro sid          comi:sid
            +--ro revision     revision

notifications:
   +---n yang-library-change
      +--ro hash    -> /modules-state/hash
```

3.2. Description

3.2.1. modules-state

This mandatory container specifies the module set identifier and the list of modules supported by the server.

3.2.2. modules-state/hash

This mandatory leaf contains the hash of the library content. The value of this leaf MUST change whenever the set of modules and submodules in the library changes. This leaf allows a client to fetch the module list once, cache it, and only re-fetch it if the value of this leaf has been changed.

If the value of this leaf changes, the server also generates a 'yang-library-change' notification.

3.2.3. modules-state/module

This mandatory list contains one entry for each YANG module supported by the server. There MUST be an entry in this list for each revision of each YANG module that is used by the server. It is possible for multiple revisions of the same module to be imported, in addition to an entry for the revision that is implemented by the server.

4. YANG Module "ietf-constrained-yang-library"

RFC Ed.: update the date below with the date of RFC publication and remove this note.

```
<CODE BEGINS> file "ietf-constrained-yang-library@2018-01-20.yang"
module ietf-constrained-yang-library {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library";
  prefix "lib";

  import ietf-comi {
    prefix comi;
  }

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

  contact
    "WG Web:   <http://datatracker.ietf.org/wg/core/>

     WG List:  <mailto:core@ietf.org>
```

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

        WG Chair: Jaime Jimenez
                  <mailto:jaime.jimenez@ericsson.com>

        Editor:   Michel Veillette
                  <mailto:michel.veillette@trilliantinc.com>";

    description
      "This module contains the list of YANG modules and submodules
      implemented by a server.

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

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

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

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

    // RFC Ed.: update the date below with the date of the RFC
    // publication and remove this note.

    revision 2018-01-20 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: Constrained YANG Module Library.";
    }

    /*
     * Typedefs
     */

    typedef revision {
      type binary {
        length "4";
      }
      description
```

```
        "Revision date encoded as a binary string as follow:
        - First byte = Century
        - Second byte = Year (0 to 99)
        - Third byte = Month (1 = January to 12 = December)
        - Forth byte = Day (1 to 31)";
    }

    /*
     * Groupings
     */

    grouping identification-info {
      description
        "YANG modules and submodules identification information.";

      leaf sid {
        type comi:sid;
        mandatory true;
        description
          "SID assigned to this module or submodule.";
      }

      leaf revision {
        type revision;
        description
          "Revision date assigned to this module or submodule.
          A zero-length binary string is used if no revision
          statement is present in the YANG module or submodule.";
      }
    }

    identity module-set {
        description
          "Base identity from which shared module-set identifiers
          are derived.";
      }

    /*
     * Operational state data nodes
     */

    container modules-state {
      config false;
      description
        "Contains information about the different data models
        implemented by the server.";

      leaf hash {
```

```
        type binary {
          length "8..32";
        }
        mandatory true;
        description
          "A server-generated hash of the contents of the library.
          The server MUST change the value of this leaf each time
          the content of the library has changed. The hash function
          and size are not specified, but shall be collision
          resistant.";
      }

      list module {
        key "sid revision";
        description
          "Each entry represents one revision of one module
           currently supported by the server.";

        uses identification-info;

        leaf-list feature {
          type comi:sid;
          description
            "List of YANG features from this module that are
            supported by the server, regardless whether
            they are defined in the module or in any
            included submodules.";
        }

        list deviation {
          key "sid revision";
          description
            "List of YANG deviation modules used by this server
            to modify the conformance of the module associated
            with this entry.  Note that the same module can be
            used for deviations for multiple modules, so the same
            entry MAY appear within multiple 'module' entries.

            Deviation modules MUST also be present in the 'module'
            list, with the same sid and revision values and the
            'conformance-type' set to 'implement'.";

          uses identification-info;
        }

        leaf conformance-type {
          type enumeration {
            enum implement {
```

```
              value 0;
              description
                "Indicates that the server implements one or more
                protocol-accessible objects defined in the YANG
                module identified in this entry.  This includes
                deviation statements defined in the module.

                For YANG version 1.1 modules, there is at most one
                module entry with conformance type 'implement' for
                a particular module, since YANG 1.1 requires that
                at most one revision of a module is implemented.

                For YANG version 1 modules, there SHOULD NOT be more
                than one module entry for a particular module.";
            }
            enum import {
              value 1;
              description
                "Indicates that the server imports reusable
                definitions from the specified revision of the
                module, but does not implement any protocol
                accessible objects from this revision.

                Multiple module entries for the same module MAY
                exist. This can occur if multiple modules import
                the same module, but specify different revision-dates
                in the import statements.";
            }
          }
          mandatory true;
          description
            "Indicates the type of conformance the server is claiming
            for the YANG module identified by this entry.";
        }

        list submodule {
          key "sid revision";
          description
            "Each entry represents one submodule within the
             parent module.";
          uses identification-info;
        }
      }
    }

    /*
     * Notifications
     */
```

```
    notification yang-library-change {
      description
        "Generated when the set of modules and submodules supported
        by the server has changed.";

      leaf hash {
        type leafref {
          path "/lib:modules-state/lib:hash";
        }
        mandatory true;
        description
          "New hash value.";
      }
    }
  }
  <CODE ENDS>
```

5. IANA Considerations

5.1. YANG Module Registry

This document registers one YANG module in the YANG Module Names registry [RFC7950].

name: ietf-constrained-yang-library

namespace: urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library

prefix: lib

reference: RFC XXXX

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

6. Security Considerations

This YANG module is designed to be accessed via the CoMI protocol [I-D.ietf-core-comi]. Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access to these data nodes.

Specifically, the 'module' list may help an attacker to identify the server capabilities and server implementations with known bugs. Server vulnerabilities may be specific to particular modules, module revisions, module features, or even module deviations. This information is included in each module entry. For example, if a particular operation on a particular data node is known to cause a

server to crash or significantly degrade device performance, then the module list information will help an attacker identify server implementations with such a defect, in order to launch a denial of service attack on the device.

7. Acknowledgments

The YANG module defined by this memo have been derived from an already existing YANG module, ietf-yang-library [RFC7895], we will like to thanks to the authors of this YANG module. A special thank also to Andy Bierman for his initial recommendations for the creation of this YANG module.

8. References

8.1. Normative References

[I-D.ietf-core-comi]
Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-02 (work in progress), December 2017.

[I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-05 (work in progress), August 2017.

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-05 (work in progress), January 2018.

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

8.2. Informative References

[I-D.dsdt-nmda-guidelines]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Guidelines for YANG Module Authors (NMDA)", draft-dsdt-nmda-guidelines-01 (work in progress), May 2017.

[I-D.ietf-netconf-nmda-restconf]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "RESTCONF Extensions to Support the Network Management Datastore Architecture", draft-ietf-netconf-nmda-restconf-02 (work in progress), January 2018.

[I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-03 (work in progress), January 2018.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-10 (work in progress), January 2018.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <https://www.rfc-editor.org/info/rfc7895>.

Author's Address

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

CoRE
Internet Draft
Intended status: Standards Track
Expires: September 4, 2018

P. Wang
C. Pu
H. Wang
J. Wu
Y.Yang
L. Shao
Chongqing University of
Posts and Telecommunications
J. Hou
Huawei Technologies
March 03, 2018

OPC UA Message Transmission Method over CoAP
draft-wang-core-opcua-transmission-03

Abstract

OPC Unified Architecture (OPC UA) is a data exchange specification that provides interoperability in industrial automation. With the arrival of Industry 4.0, it is of great importance to implement the exchange of semantic information utilizing OPC UA Transmitting in CoAP. This document provides some transmission methods for message of OPC UA over CoAP.

Table of Contents

1. Introduction 2
   1.1. Conventions and Terminology 3
2. Overview of OPC UA 3
   2.1. Protocol Stack 3
   2.2. Request/Response Model 5
3. Specification of OPC UA over CoAP 6
4. Transmission scheme 7
   4.1. Direct transmission 7
   4.2. REST transmission for OPC UA 8
5. Publish subscription for OPC UA over CoAP 9
6. Use Cases of OPC UA over CoAP 9
   6.1. Factory data monitoring based on web pages 9
   6.2. Offline/Online diagnostic system for resource-constrained factories 10
   6.3. Factory data analysis based on cloud 11
7. Security Considerations 11
8. IANA Considerations 11
9. References 11
   9.1. Normative References 11
   9.2. Informative References 12
Authors' Addresses 13

1. Introduction

Internet of things is one of the attractive applications for CoAP [RFC7252]. Utilizing OPC UA [IEC TR 62541-1] Transmitting over CoAP could meet the demand for industry 4.0 based on the exchange of semantic information [I-D.wang-core-opcua-transmition-requirements]. In resource-constrained scenarios, OPC UA can effectively use energy, improve productivity and shorten the product manufacturing cycle by

building information model and using its cross-platform characteristic. Similar to OPC UA, CoAP message is exchanged in server/client mode. However, both of them have specific clients and servers. Driven by this, to implement OPC UA Transmitting over CoAP, the main problem to be solved is how OPC UA packets are transmitted over CoAP. For the transport layer of OPC UA, the main message transmission method is TCP or HTTP. It is worth noting that the design of CoAP is inspired by HTTP, thus, there are some similarities in transmission method between them. This document provides some transmission methods for message of OPC UA over CoAP, so that the communication between OPC UA client and OPC UA server could be established.

1.1. Conventions and 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].

OPC: OLE for Process Control

OPC UA: OPC Unified Architecture

SOAP: Simple Object Access Protocol

REST: Representational State Transfer

HMI: Human Machine Interface

2. Overview of OPC UA

OPC Unified Architecture (OPC UA), standardized as IEC 62541, is a client-server communication protocol developed by OPC Foundation for safety, reliable data exchange in industrial automation. It is the evolution product of OPC (OLE for Process Control, where OLE denotes Object Linking and Embedding), the widely used standard process for automation technology, and is of great importance in realizing industry 4.0. By introducing Service-oriented architecture (SOA), OPC UA enables an open, cross-platform communication with the advantages of web services, robust security and integrated data model.

2.1. Protocol Stack

OPC UA is an application layer protocol that can be built on existing layers 5, 6 or 7 protocols such as TCP/IP, TLS or HTTP. The

OPC UA application layer consists of four sublayers: UA Application, Serialization Layer, Secure Channel Layer and Transport Layer (see Figure 1).

Serialization Layer includes two kinds of data encoding methods: UA Binary and UA XML. The UA XML, based on SOAP/HTTP or SOAP/HTTPS, is firewall friendly. On the other hand, the UA Binary, with least overhead and resource cost, offers an optimized speed and throughput.

The security layer varies according to the selected encoding format. For the HTTPS-based situation, security is implemented at TLS but Security Channel should still be presented even empty. It is worthwhile noting that the communication based on SOAP/HTTP has been deprecated since 2015, due to the lack of industrial approbation in the WS Secure Conversation.

For the transport layer (not the layer in OSI 7 layer model), options can be UA TCP, HTTPS, SOAP/HTTPS, and SOAP/HTTP. OPC UA defines a UA TCP protocol, which differs from HTTP in two main features: the allowance of responses to be returned in any order and to be returned on a different TCP transport end-point. In addition, UA TCP defines the interaction with the upper security channel.

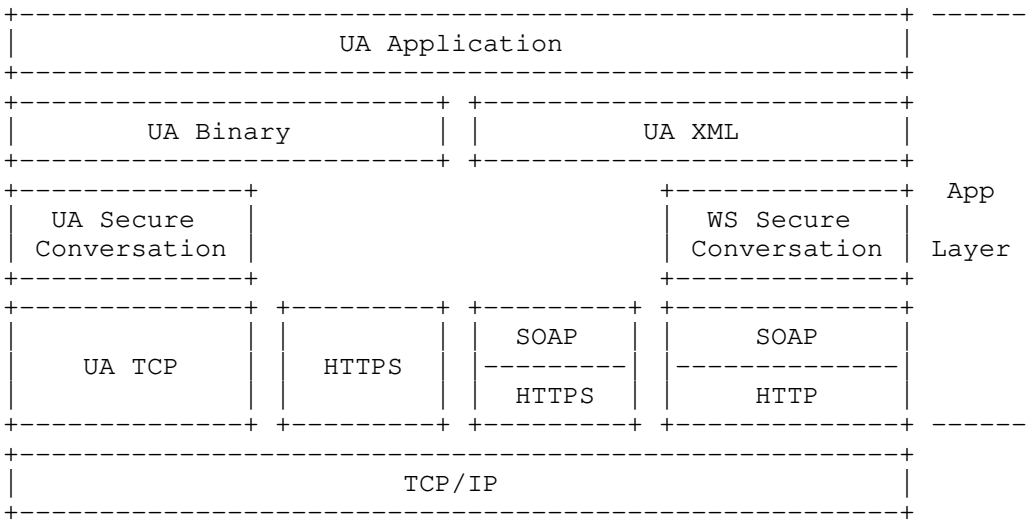


Figure 1: Layering of OPC UA over TCP/IP

2.2. Request/Response Model

The message exchange in UA binary mode is illustrated in Figure 2. After opening the socket, the client starts the connection with the server by using "hello" (HEL) and "acknowledge" (ACK) messages. Afterwards, a pair of messages is needed to open the security channel and define the encryption property. Then another two pairs of messages are exchanged so as to create and activate a session between the client and the server respectively. After these steps, the connection is initiated and the client can send request messages for services. When the request/response process is finished, a reverse process is required for disconnection.

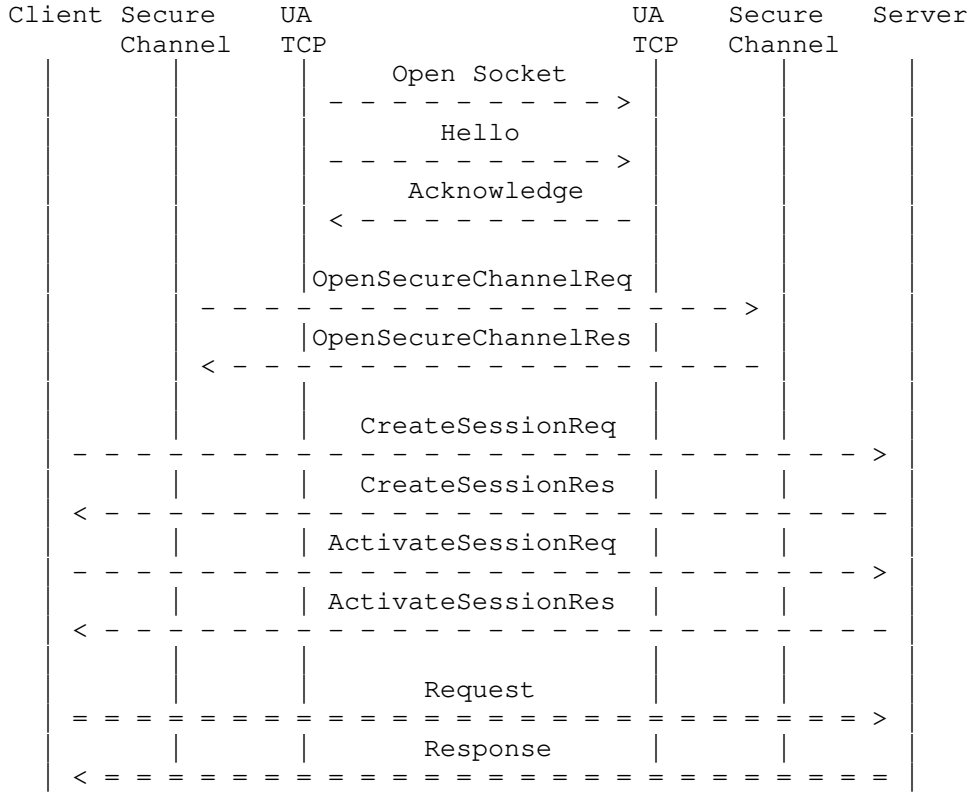


Figure 2: Request/Response Process of UA TCP

3. Specification of OPC UA over CoAP

As mentioned in section 2.1, OPC UA communications can be conducted through four options, among which two are related to HTTPS: HTTPS => UA Binary; HTTPS => SOAP => UA XML.

Constrained Application Protocol (CoAP) is an application layer protocol for constrained nodes and networks, which is designed to easily translate to HTTP for integration with the web. Although CoAP is built on the unreliable transport layer UDP, it offers a security mode binding to Datagram Transport Layer Security (DTLS). This document proposes a transmission scheme based on CoAPs (CoAP + DTLS) for constrained scenarios. The transmission based on CoAP over Transport Layer Security (TLS) is available [RFC8323].

The protocol stack of the CoAP based OPC UA is illustrated in Figure 3, including two options at Serialization Layer: UA Binary and UA XML. OPC UA packets are encoded in either binary or xml format, and the option field in the CoAP header can specify parameters that support both formats. Therefore, according to the format specified by the CoAP header, the entire packet of the OPC UA can be encapsulated in the payload of the CoAP message for direct transmission.

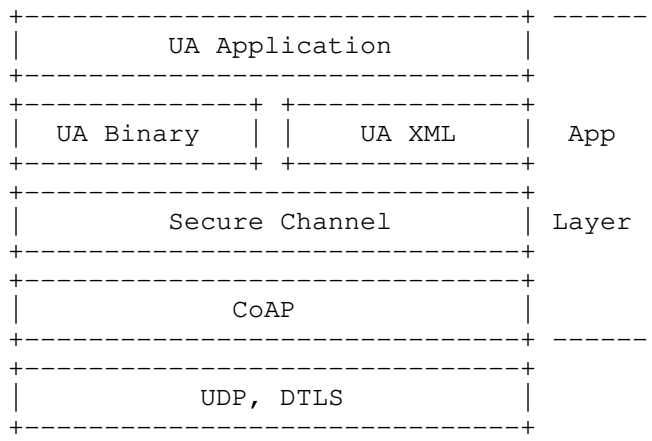


Figure 3: Layering of OPC UA over UDP

Both binary and XML encoding modes are based on the CoAP with an empty UA secure channel in between. For the XML encoding mode, since CoAP layer supports XML encoding format, the SOAP layer in the original stack is not needed.

4. Transmission scheme

4.1. Direct transmission

The transmission of OPC UA supports TCP protocol and HTTP protocol. CoAP is seen as a simplified HTTP protocol so that it can be applied to resource-constrained network. Therefore, this document considers the use of CoAP to directly transfer OPC UA messages. OPC UA packets are encoded in either binary or xml format, and the optional fields in the CoAP header specify parameters to support these two formats. Therefore, according to the format specified by the CoAP header, the entire packet of the OPC UA can be encapsulated in the payload of the CoAP message for direct transmission, as shown in Figure 4. According to CoAP, noted that this method of transmission needs to be modified on the server side and the client side of the OPC UA according to CoAP.

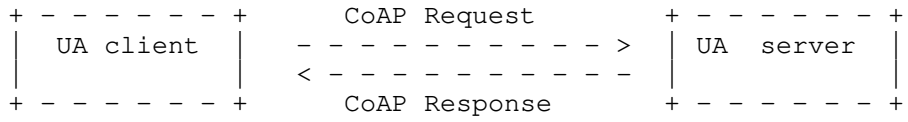


Figure 4: Direct transmission OPC UA based on CoAP

For supporting HTTP, a CoAP proxy can be established between OPC UA client and OPC UA server.

As shown in Figure 5, assuming all OPC UA servers are based on CoAP, and all OPC UA-CoAP servers can be considered to form a constrained network, then introducing a UA-to-CoAP proxy at the boundary of the network. When a traditional OPC UA client initiates an HTTP request to the UA-CoAP servers which is in the constrained network mentioned above, the UA-to-CoAP proxy maps the http request to the corresponding CoAP request and sends it to the UA-CoAP server in the network. After receiving the request, the UA-CoAP server sends a response to the UA-CoAP proxy. The proxy maps the CoAP response to the HTTP response and returns it to the UA client. For the UA client, the network proxy and conversion are transparent, in this way, the transfer of OPC UA in CoAP does not need to make any changes to the UA Client.

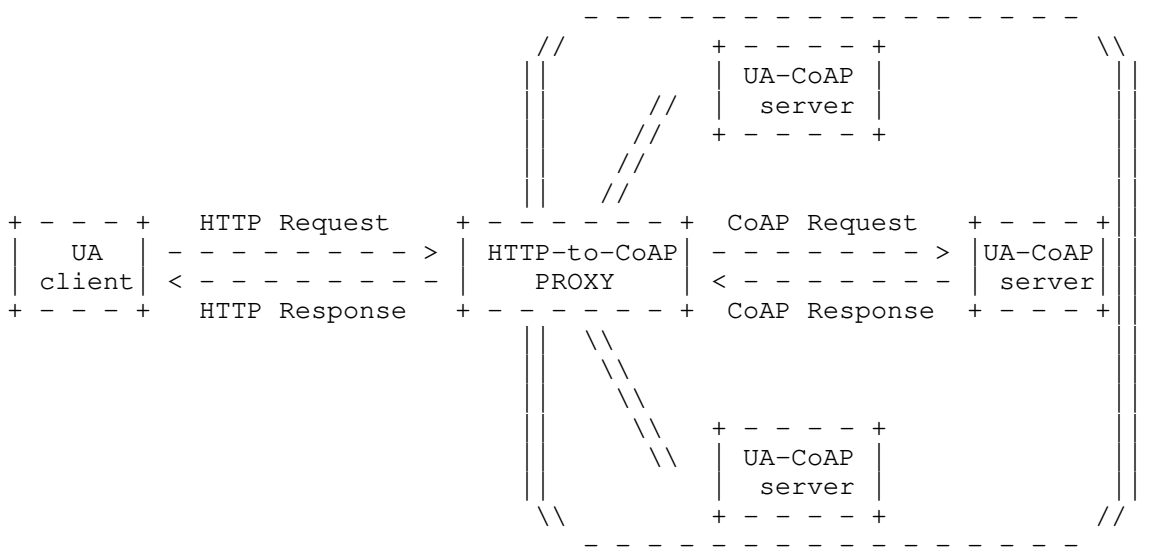


Figure 5: Proxy for OPC UA to CoAP

4.2. REST transmission for OPC UA

OPC UA is a set of data which exchange specifications for industrial communication, the core of the OPC UA protocol are information modeling and transmission, which marks each node in the address space with a unique identifier. A series of state interactions are needed before performing normal reading and writing, including message handshaking, opening a secure channel, creating a session, activating a session, etc. Besides, some states also need to be maintained during read and write operations.

In OPC UA, each node has an independent identifier in the address space, and different types of nodes can establish contact with each other by referencing. OPC UA defines a variety of services, and these services are fixed, because of this, the users cannot modify OPC UA services according to their own ideas. In general, services in OPC UA cannot be considered stateless, but many of them which are also commonly used are inherently stateless, e.g. FindServers, Read, Write [RICO]. The above features are in line with the REST architecture, due to CoAP is based on the REST architecture. Therefore, it is possible to simplify the interaction before the OPC UA performs the normal communication, and carry the OPC UA message by using the communication mode of the CoAP. Communication process is shown in Figure 6.

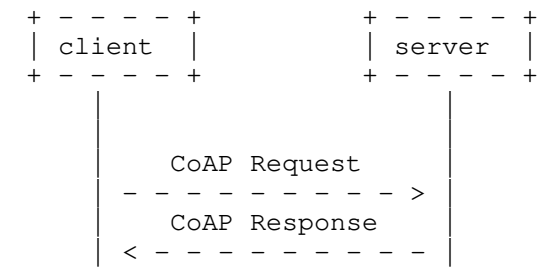


Figure 6: REST architecture communication of OPC UA

In Figure 2, the traditional OPC UA requires a series of interactions between normal read and write operations. Figure 6 shows that when using CoAP to carry OPC UA message, the interaction process is significantly reduced, which is conducive to the application of OPC UA in the restricted scenes. The cost of simplifying the interaction process is that the secure channel number is set to 0 by default, how to conduct secure data interaction needs further discussion.

5. Publish subscription for OPC UA over CoAP

As an application sublayer, CoAP provides publish-subscribe functionality, primarily for resource or network-constrained scenarios. Introducing proxy into the network [I-D.ietf-core-coap-pubsub], when a node needs to sleep, the node information is sent to the proxy agent, when another node requests to obtain information of this node, the broker release function can provide information. OPC UA defines the publish-and-subscribe function as a service in the service set. The client initiates the subscription request directly to the server, and the server periodically sends the information to the client. Comparing the characteristics of the two protocols, it is found that each of them has its own advantages. Joint design can be conducted for constrained applications.

TODO.

6. Use Cases of OPC UA over CoAP

6.1. Factory data monitoring based on web pages

Description: At present, the monitoring and management systems of the factory mostly exist in the form of software on the PC and the mobile. The drawback is that when the whole factory system is needed to upgrade, the monitoring software must be upgraded as well. It may cause the huge workload that will bring the time and financial

burden on the factory. CoAP is a HTTP-like communication protocol designed specifically for resource-constrained environments so that can be used in the factory because the sensor nodes in the factory mostly are resource-constrained. CoAP can easily transform to HTTP and OPC UA can consolidate the different protocols in the plant by building a unified information model.

Goal: PC and mobile devices can check and monitor the data by visiting WEB pages after CoAP is converted to HTTP. Avoiding large-scale software upgrades caused by system upgrades, while also reducing the development of mobile software, thereby reducing factory costs.

Requirements: the OPC UA information model should be encapsulated into CoAP data load. Because of the capacity limitation of UDP packet (MTU is 1472 bytes), in some cases, it is needed to compress, fragment, and reassemble packets.

6.2. Offline/Online diagnostic system for resource-constrained factories

Description: There are two modes existing in the factory's self-diagnosis system, the offline mode and the online mode. In the offline mode, the self-diagnostic device could use getHistorical, a service from OPC UA, to get historical Data. In the online mode, Both OPC UA and CoAP support pub/sub so that the monitoring system can obtain the data from a specific device in a short reaction time to determine its operating status. CoAP, as a resource-constrained factory transmission protocol, can easily access many web services APIs, add functionality that the factory can implement and let the system have a certain degree of expansibility. OPC UA could create a unified information model that realizes factory interoperability and protocol uniformity.

At same time, the controller node can diagnose and regulate other nodes by receiving their data rather than transferring them to HMI (The M2M Communication). Generally, using UDP is the best choice, however, CoAP's UDP not only has excellent stability but also has relatively few packet loss rates. The unified model of OPC UA enables all nodes to communicate without obstacles.

Goal: Using OPC UA over CoAP to enable factory offline history data diagnostics, online real-time monitoring, publish subscriptions and Achieving network nodes M2M communication.

Requirements: OPC UA uses SOA architecture, while CoAP uses REST architecture, it is necessary to design a reasonable architecture for OPC UA over CoAP.

6.3. Factory data analysis based on cloud

Description: Currently, there are many clouds (AWS, Windows Azure, etc.) which have different kinds of APIs. These clouds could achieve machine learning, data-flow analysis and so on for factory's data. Using CoAP can effectively access these interfaces and fully take advantage of clouds capabilities. At present, many factories have begun to use the cloud to improve production status, So the biggest benefit to use CoAP in factories is that CoAP could let devices to use cloud's applications in resource-constrained factories so that to achieve intelligent control. OPC UA can consolidate the different protocols in the plant by building a unified information model. Based on the content mentioned above, the field devices in the factories can transfer their data directly and immediately to the cloud without sending them to border routers or HMI.

Goal: Using OPC UA over CoAP to transfer field devices' data to the cloud.

Requirements: Using OPC UA to modeling the different types of data in the plant and then using CoAP to directly transfer the factory's data to the cloud.

7. Security Considerations

This document does not add any new security considerations beyond what the referenced technologies already have.

8. IANA Considerations

This memo includes no request to IANA.

9. References

9.1. Normative References

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol", RFC 7252, June 2014, <https://tools.ietf.org/html/rfc7252>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997, <https://tools.ietf.org/html/rfc2119>.

[RFC8075] Castellani, A., Loreto, S., and A. Rahman, "Guidelines for HTTP-to-CoAP Mapping Implementations", RFC 8075, November 2016, <https://tools.ietf.org/html/rfc8075>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC8323, February 2018.

9.2. Informative References

[IEC TR 62541-1]
IEC, "OPC unified architecture-Part1:Overview and concepts-IEC 62541", 2016, <https://webstore.iec.ch/preview/info_iec62541-1%7Bed2.0%7Den.pdf>.

[I-D.wang-core-opcua-transmition-requirements]
Wang, H., Pu, C., Wang, P., Yang, Y., and D. Xiong, "Requirements Analysis for OPC UA over CoAP", draft-wang-core-opcua-transmition-requirements-02 (work in progress), December 2016.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol(CoAP)", draft-ietf-core-coap-pubsub-03 (work in progress), February 2018.

[RICO] Gruner, Sten, Julius Pfrommer, and Florian Palm. "RESTful industrial communication with OPC UA." IEEE Transactions on Industrial Informatics 12.5 (2016):1832-1841. <http://ieeexplore.ieee.org/abstract/document/7407396/>

Authors' Addresses

Ping Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: wangping@cqupt.edu.cn

Chenggen Pu
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: mentospcg@163.com

Heng Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6248-7845
Email: wangheng@cqupt.edu.cn

Junrui Wu
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-18580183135
Email: wjr930914@gmail.com

Yi Yang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: 15023705316@163.com

Lun Shao
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: yjsslcqupt@163.com

Jianqiang Hou
Huawei Technologies CO.,LTD
101 Software Avenue,
Nanjing 210012
China

Phone: (86)-15852944235
Email: houjianqiang@huawei.com